An Engineering-Oriented Formal Framework for Railway Interlocking Systems Requirements Specifications

Von der

Fakultät Architektur, Bauingenieurwesen und Umweltwissenschaften

der Technischen Universität Carolo-Wilhelmina

zu Braunschweig

zur Erlangung des Grades einer

Doktoringenieurin (Dr.-Ing.)

genehmigte

Dissertation

von

Yuen Man Hon

geboren am 11.6.1980

aus Hongkong

Eingereicht am	13. Mai 2009
Disputation am	11. September 2009
Berichterstatter	Prof. Dr. -Ing. Jörn Pachl
	Prof. em. Dr. H.-D. Ehrich

2009

To Daisy and Frank who
allow me to try and question everything.

Acknowledgement

This research work was achieved during the time as a research assistant at the Institute of Railway Systems Engineering and Traffic Safety (IfEV), Technische Universit"at Braunschweig. It was funded by the Karl-Vossloh-Stiftung. Therefore, I would like to first express my gratitude to this foundation. Secondly, I am highly thankful to Prof. Dr. Pachl and Prof. Dr. Ehrich for their support and useful suggestions throughout this research work.

I own my deepest gratitude to one of my colleagues from IfEV, Dr. Jan-Tecker Gayen. His patience to me, continuous guidance, help and great care enabled me to complete this thesis. I will never forget those delightful discussions and the things that he taught me. I also wish to thank other colleagues from IfEV, particularly Hannes Franzmeyer and the "Stellwerk Truppe" (with Friederike Salbert, Henning Blum and Katja May). My special thanks must also be dedicated to one of the members, Friederike Salbert who is always willing to help and listen to me without hesitation.

I am as ever indebted to my parents, Tony Hon and Daisy Ma for their love and support throughout my life. I would also like to thank my siblings Brian Hon and Muriel Hon for their deep affections, especially my adorable brother; he has been always very much supportive and given me good suggestions since the beginning of my study. I am also grateful to my very good friend Sibylle Simmerling, who cares about me and shares experience with me. Finally, this thesis and my work would not have been possible without the indulgence, approvals and encouragements of my husband, Frank Klawonn.

Contents

List of Figures

Abstract

In the German railway domain, computer-based interlocking system requirements specifications are written in natural language. It is well-known that natural language written specifications are easily misinterpreted because of their possible unclearness and ambiguity. Furthermore, checking the correctness of these specifications is difficult. Imprecise and incorrect system requirements can reduce the efficiency of a system development process. They might also lead to errors in the system. The costs of amending these errors become higher as the development process proceeds. For safety critical systems like interlocking systems, the highest costs are then the loss of human life. Using formal methods to specify these system requirements in the early stage of a development process is the proposed solution to solve the above stated problems. Because of the advantages of applying and the suggestions from CENELEC to use formal methods for railway system development, a number of research projects in this topic have been carried out. The results of these projects have shown that requirements of formal specifications are precise. Their correctness can be checked and increased. However, to apply these formal methods, one needs a strong mathematical background. The involvement of railway engineers in formally specifying, analyzing and verifying requirements is quite limited in these projects. As a result, formal methods are not commonly used in the railway domain. The main goal of this work is to develop an engineering-oriented formal framework, such that not only computer scientists, but also railway engineers can benefit from the ideas of formal methods. The developed formal framework is called Object-Based Lastenheft (OBLH). This framework is composed of well-defined mathematical concepts and ideas for specifying, analyzing and verifying interlocking system requirements. A tool called the OBLH tool is also implemented, such that mathematical operations of these concepts can be executed automatically. This framework is based on applications of propositional logic and decision tables. In this framework, railway engineers can specify system requirements formally in both propositional logic and decision tables. Specifications become clear and unambiguous. The requirements in propositional logic can be transformed to decision tables, such that they can be easily analyzed by different professions. Once requirements are specified formally, computer scientists or railway engineers can carry out analysis and verification of the specifications based on the defined methods in the OBLH tool. This formal

framework was successfully applied in an industrial project. It was shown that railway engineers played an active role in using this formal framework. The formally specified system requirements become precise and easy to be understood. Furthermore, the correctness of these requirements is also increased. In conclusion, OBLH is an engineering-oriented formal framework which can successfully bring the advantages of using formal methods to different professions. The product of this framework is a clear, explicit and correct interlocking system requirements specification. Clear and correct system requirements reduce the costs of system developments and the safety of the systems is also increased.

Chapter 1

Introduction

One of the functions of a railway interlocking system (RIS) is to establish safe routes. No train collisions can happen when scheduled trains are driven along these safe routes. In the German railway domain, most of the interlocking systems are realized by computer programs. The functionalities of these computer-based interlocking systems are written in documents, interlocking system requirements specifications (e.g. LH-ESTW-R [Tut06]). System requirements in these specifications are implemented by system development teams that are composed of experts from different professions, like railway engineers and computer scientists. These specifications share some similar characteristics. They are written in German natural language. Their system requirements are expressed and elaborated by using complex railway scenarios and concepts. These complicated railway concepts are not easy to be understood and implemented by team members, the efficiency of the system development might then be affected. Furthermore, it is well-known that system requirements written in natural language can cause misinterpretations. Their correctness is also difficult to be checked. Incorrectness in requirements can result in errors in the output of each development phase. More resources are needed to amend these errors if they are found in a later stage of the development process. In the worst case, these errors might cost life of humans. Therefore, for a safety critical system, like RIS, ensuring the clarity and correctness of its system requirements is an important task.

One way to handle these problems is to apply formal methods in interlocking system development processes. In formal methods, system requirements must be expressed by the defined mathematical languages. Via the well-defined syntax and semantics of the languages, unclearness and misinterpretations can be avoided. Once requirements are formally defined based on the specification languages, mathematical concepts can be applied to analyze the correctness of these requirements. Programs can then be implemented to carry out the mathematical operations of these concepts automatically. The CENELEC railway application standards have also addressed the importance of using formal methods for developing railway systems and the assessment of the safety integrity levels of these systems is affected by the formality of their development processes [CEN99, CEN00b, CEN00a]. Because of the advantages of using formal methods and suggestions from CENELEC,

the feasibility of applying formal methods to interlocking system development has been investigated in a number of research projects.

These research projects have shown that incorrectness of system requirements can be found and the clarity of specifications also increases. However, railway engineers play a passive role in the whole formal system development process. Without the corresponding mathematical background, railway engineers have no chance to specify the system model formally. System models and checking conditions for verification are mostly defined by formal experts and no assistance is provided for railway engineers to understand these mathematical models or conditions in these projects. The correctness of their definitions depends entirely on the formal experts' understanding of the railway domain. Confidence of railway engineers in these definitions and the corresponding verification results are therefore decreased. As a result, applications of formal methods have not become popular among railway engineers.

These observations have motivated this research work. The main goal of this work is to develop an engineering-oriented formal framework, such that different professions of interlocking system development teams can benefit from the ideas of formal methods. Moreover, the framework should be designed to be used in the early stage of the development process, such that unclearness and inconsistent requirements will not be brought to the next phase of the system development process. This framework should be composed of mathematically well-defined concepts and ideas for specifying, analyzing and verifying interlocking system requirements. Because of the formal definitions of these concept, a tool can then be implemented, such that these concepts can be executed automatically. Support for railway engineers to understand formally specified requirements must be provided. They must also able to specify and check the correctness of these formal requirements with the help of the developed tool alone.

In Chapter 2, a general introduction to German railway interlocking systems is first given. The characteristics of their specifications and the problems that are caused by these characteristics are also discussed in this chapter. In the next chapter, formal methods and their applications to the railway domain are then introduced and the mentioned motivation is further elaborated. In Chapter 4, the concepts and the tool that have been developed to achieve the goals of this work are explained informally. The mathematical background and the concepts of the development framework are introduced and formally defined in Chapter 5. The discussion and conclusion of this work are given in Chapter 6 and Chapter 7, respectively.

Chapter 2

Railway Interlocking Systems and Interlocking System Requirements Specifications

Before a train arrives at or departs from a railway station, the railway interlocking systems (RIS) is responsible for setting up safe routes (German: Fahrstraße) based on safe route requests. It ensures the proper usage and setting of each infrastructure element along the safe route. Safe routes ensure that no train collision would happen and the trains can be driven from the planned starts to the planned destinations. Interlocking system requirements specifications, like [Tut06], specify requirements that need to be implemented in a German regional computer-based interlocking system, such that development of safe routes can be guaranteed. Therefore, the system development teams must interpret the specifications correctly, such that the requirements can be properly implemented. However, most of the interlocking system requirements specifications are written in natural languages. These system requirements are specified with certain characteristics. These two properties might lead to some problems to the system development.

In this chapter, an introduction to railway stations and infrastructure elements is given in Section 2.1.1 and 2.1.2, respectively. The principles of interlocking systems are then discussed. The final section focuses on the characteristics of interlocking system requirements specifications and the corresponding problems that might be brought to the system development teams.

2.1 Railway Stations and Infrastructure Elements

As mentioned above, an RIS establishes safe routes by coordinating the setting and employment of infrastructure elements of railway stations, such that safe movements of every train through a station are guaranteed. In order to understand the definition and the principles for developing safe routes, one has to know the basic components, called infrastructure elements, of a railway station. Different countries have different kinds of names, definitions and philosophies to describe their railway systems. The focus of this work is put on the German practice only.

In this section, railway stations and infrastructure elements that have been considered in this work are introduced and described.

2.1.1 Railway Stations

Figure 2.1 shows the layout of a railway station. A railway station is located within the home signals limits (German: Bahnhof) in German railways [Pac04a, Pac04b]. The home signals limits are defined by a pair of opposite home signals, that are A and F in Figure 2.1. They consist of minimally one turnout for diverting trains from one track to another (see 2.1.2). Trains might arrive, depart and pass within the home signals limits. The interlocking system coordinates settings of the infrastructure elements within the home signals limits to ensure safe movements of trains that might pass through or stop at the station. The main tracks that are located outside the home signal limits are called open line.

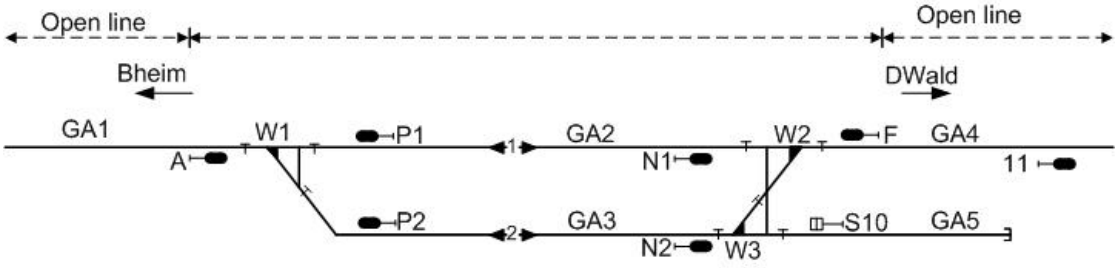


Figure 2.1: The layout of railway station CSStadt

2.1.2 Infrastructure Elements

There are different kinds of infrastructure elements that are located within the home signals limits. The type of infrastructure elements that have been considered in this work are the following ones:

- Tracks: The basic infrastructure components are tracks (German: Gleis) that trains run on. Tracks can be further classified into main tracks and sidings based on the role they play in the movements of railway vehicles. The authorized movements, called train movements, of railway vehicles are only allowed on main tracks, while making up a train, called shunting movements, are undergone only on sidings.
  In Figure 2.1, main tracks are GA1, GA2, GA3 and GA4. The direction of traffic of each track is specified by arrows. For example, in Figure 2.1, bi-directional traffic is allowed on all tracks. This means a train from Bheim can stop on the track GA2 at the signal N1, while a train from DWald can stop on the track GA2 at the signal P1. These two events should not happen at the same time, otherwise a collision would take place.

- Points: Instead of moving only straight ahead, trains can also turn to another track. For this purpose, there are different kinds of track arrangements:

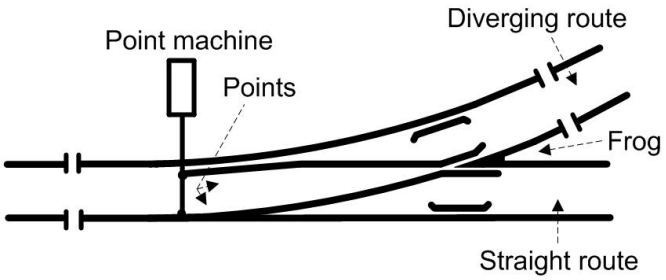


Figure 2.2: A turnout

turnouts and crossings. As mentioned in Section 2.1, home signal limits consist of at least one turnout. A turnout allows trains to move from one track to another. A turnout (see Figure 2.2) is composed of movable points (German: Weiche), rails and a frog. A train can divert to another track when the points are set in a proper position such that the train runs into the correct track. In Figure 2.3, in order to develop a route from A to B or B to A, the point is set to the direction of the straight track (German: Stammgleis) and from A to C or C to A, the point is set to the direction of the diverging track (German: Zweiggleis). Furthermore, the vehicle movement from A to B or A to C is called a facing point movement (German: Spitz befahren). The vehicle movement from B to A or C to A is called a trailing point movement (German: Stumpf befahren).

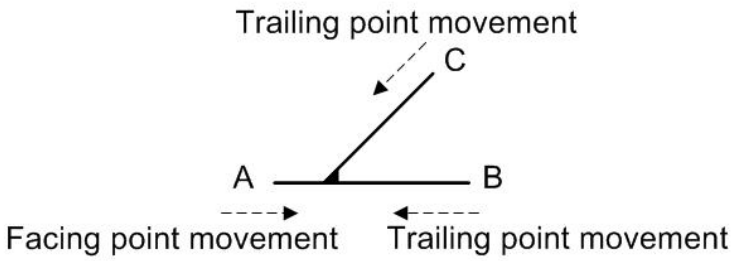


Figure 2.3: A point

Another kind of track arrangement that can be found in home signal limits is a crossing (German: Kreuzung). It allows trains to cross other tracks at an angle [Bon01].

- Signals: Signals and tracks are called track elements. Signals are important components that are used to control the train or shunting movements and control the speed of trains, such that no collision would take place. There are different kinds of signals within the home signals limits: Home signals,

exit signals and intermediate interlocking signals. Home signals are used to indicate the starting and ending point of the home signals limits and guide the train to enter the home signals limits. In Figure 2.1, the home signal A directs trains that are driven from Bheim to CSStadt while signal F directs trains coming from DWald to CSStadt. Exit signals are responsible to guide trains to leave the home signal limits to the open line. Intermediate interlocking signals are those signals that are neither home nor exit signals within the home signal limits. Many railways use three signal aspects: stop, approach and clear. When a signal indicates a stop aspect, drivers must stop the train. With an approach aspect, drivers should prepare to stop at the next signal, in other words, the driver should start to reduce the speed of the train. Finally, clear aspect means that the driver can drive into the track behind this signal.

2.2 Railway Interlocking Systems

An RIS has a number of functionalities. One of the functions is displaying the status of safe routes development and the conditions of each infrastructure element of the railway station on the Graphical User Interface (GUI) of the interlocking system (German: Anzeige der Fahrdienstleiterarbeitsoberfläche). Another important function is developing, monitoring and releasing safe routes within a railway station, such that trains can reach their destinations safely. An RIS consists of an interlocking logic and an interlocking machine. The realization of the principles of interlocking is called the interlocking logic (German: Fahrstraßenlogik). In other words, this interlocking logic needs to be implemented in the RIS, such that a safe route can be properly developed, monitored and released. An interlocking machine is used to control and interlock the track elements to ensure that they function properly for the safe route.

In this section, the definition of a safe route is first given. Then, interlocking logics and interlocking machines are briefly described.

2.2.1 Safe Routes

A safe route is composed of a route (German: Fahrweg), an overlap (German: Durchrutschweg/DWeg) and the flank protections (German: Flankenschutz) for the route and overlap (see Figure 2.4). The infrastructure elements that are part of the safe route must be set properly. They are locked as elements for this safe route based on their function provided for the safe route, such that the development of conflicting routes can be avoided.

The components of a safe route are further discussed as follows:

- Route: The route is used by the train to move from the planned start to the planned route destination (German: Ziel). Elements that are located between the start and the destination of the route are reserved and locked as route elements (German:Fahrwegelement). Their proper setting for safe

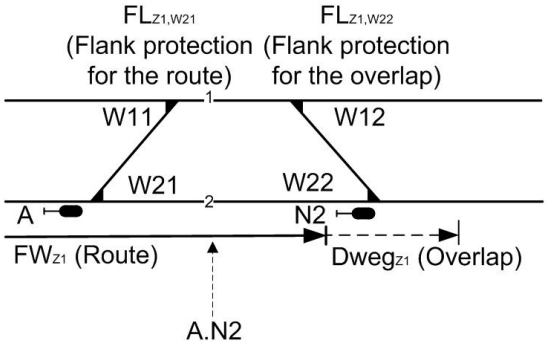


Figure 2.4: A safe route

routes is also marked during the reservation. The RIS checks the setting and marking of elements, when elements are requested to be used by another requested safe route. For example, to develop the route A.N2[1], the point W21 must not be reserved or locked for other safe routes and is put to the direction of the straight track. In Figure 2.4, the route begins at the signal A and ends at the signal N2. The train director (German: Fahrdienstleiter) inputs the command A.N2 into the RIS, such that the RIS will start to develop this requested safe route. FW_{Z1} indicates the route of this requested safe route. The route elements of FW_{Z1} are the point W21, the signal A, signal N2 and the track section between the signals. During the development of FW_{Z1}, W21 is reserved and is marked as the direction of the straight track because this is the proper setting of W21 for FW_{Z1}. If elements are locked or reserved for a safe route as route elements, they must be used by other safe routes as route elements. Otherwise, collision might happen when a route element is locked or reserved for more than one route as route element and be used at the same time.

- Overlap: The overlap is a track section that is in front of the route. This track section must be ensured to be clear, such that if the driver does not stop the train in time, the train will not collide with a train ahead. Elements along the overlap are locked as overlap elements, for example, W22 is locked as an overlap element for $DWeg_{Z1}$ in Figure 2.4.

- Flank protection: Flank protection ensures that no trains can be driven into the safe route through points. This means, RIS needs to search flank protections for points that belong to the route and overlap. An element that provides flank protection to a route element or overlap element is called a flank protection element (German: Flankenschutzelement). Signals and points can be flank protection elements. The points that provide flank protections need to be set properly, such that trains cannot run into track sections of the safe

[1]X.Y is the RIS input command of a safe route development. X indicates the start and Y indicates the requested route destination.

route. For example, in Figure 2.4, W11 provides a flank protection $FL_{Z1,W21}$ to W21, while the flank protection $FL_{Z1,W22}$ of W22 is provided by W12. W11 and W12 need to be set to the straight position, such that trains on track 1 cannot run into track 2.

As mentioned in Section 2.1.1, trains might arrive, depart and pass through a railway station. Safe routes can be classified into three categories. They are safe entrance routes (German: Einfahrstraßen), exit safe routes (German: Ausfahrstraßen) and non-stop safe routes (German: Durchfahrstraßen).

- Safe entrance routes (German: Einfahrstraßen): When a train is scheduled to stop at the railway station, a safe entrance route is requested to be established before its arrival. A safe entrance route is composed of a route, an overlap and the flank protection for the route and overlap. The route begins at a home signal and ends at an exit signal within the home signals limits. For example, in Figure 2.6, if a train comes from the direction of Bheim and is scheduled to stop in front of N1, then a safe entrance route is composed of FW_{Z1}, $DWeg_{Z1}$, $FL_{Z1,W1}$ and $FL_{Z1,W2}$.

- Safe exit routes (German: Ausfahrstraßen): An safe exit route is established for the train to depart the railway station and be driven to the open line. It is composed of a route and the flank protection for this route. An safe exit route begins at an exit signal and ends at the interface. This interface separates the home signals limit and open line is called interface Bf/Str (German: Schnittstelle Bahnhof/Strecke (Bf/Str)). Interface Bf/Str is normally located between the last point of the railway station and the beginning of the open line (see Figure 2.5). RIS needs to ensure the block systems' requirements have been fulfilled in the track section between the interface Bf/Str and the first block signal. For example, in Figure 2.5, this track section is the space between the interface Str/Bf and Signal 11. Ensuring the safe train separation in open lines is guaranteed by automatic block systems (German: Streckenblock) in most of the German track sections.

- Non-stop safe routes (German: Durchfahrstraßen): A non-stop safe route is composed of a safe entrance route and an safe exit route. In Figure 2.6, the non-stop safe route is composed of the safe entrance route (FW_{Z1},$DWeg_{Z1}$, $FL_{Z1,W1}$ and $FL_{Z1,W2}$) and the safe exit route (FW_{Z2} and $FL_{Z2,W2}$). A non-stop safe route can also be established by requesting the safe exit route and safe entrance route separately or together.

2.2.2 Interlocking Logic

The principles of interlocking state that all movable elements within a safe route must be set and locked properly, conflicting routes must be locked and the tracks must be clear before issuing the safe route to a train. An interlocking logic is

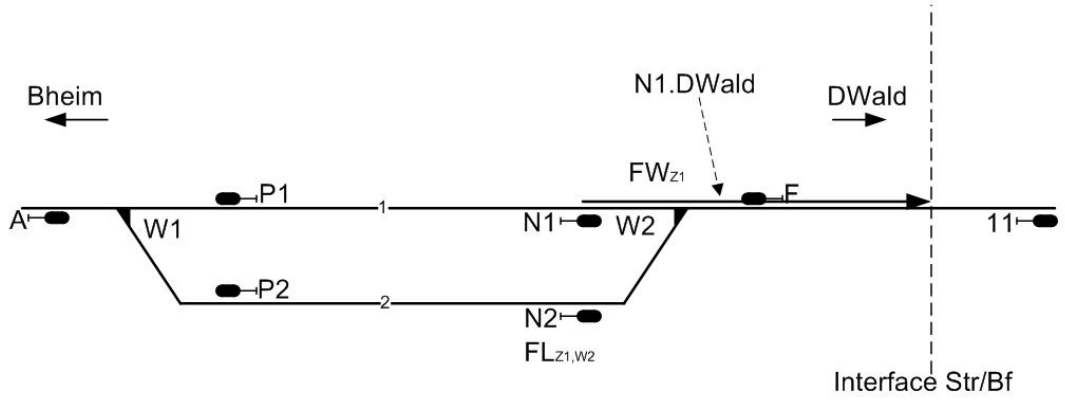


Figure 2.5: An safe exit route

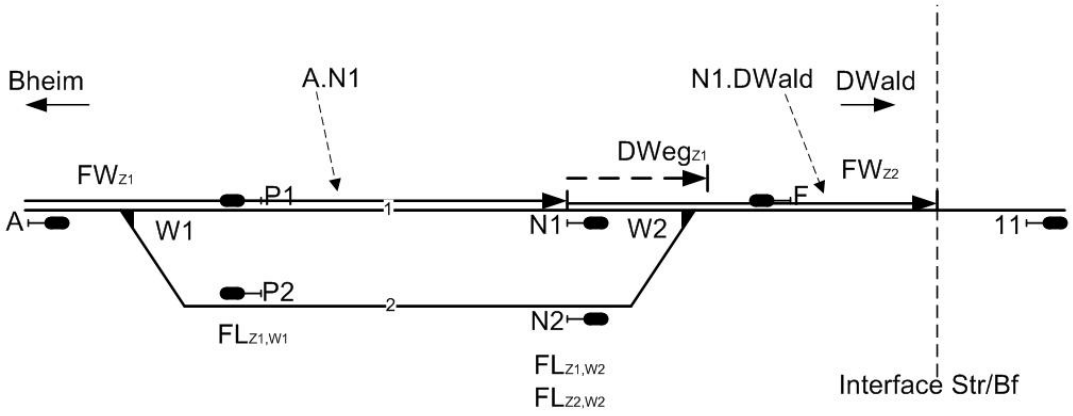


Figure 2.6: An non-stop route

defined to achieve these goals. In German computer-based interlockings, a number of procedures and statuses of safe routes are defined to achieve these goals. For example, the possibility to develop a requested safe route is checked in a procedure called Zulassungsprüfung Zugstraße. In this thesis, it is named as ZPZ and it will not be translated into English. The possibility of reserving or locking each of the infrastructure elements along the safe route is analyzed. The usage of each infrastructure element within the requested route and overlap is checked based on defined conditions called ZPZ conditions (German: ZPZ-Bedingungen). One of these conditions is that the requested route element must not be reserved or locked for another route. One of the purposes of ZPZ is to avoid setting up conflicting routes. If the elements within the requested safe route fulfill all ZPZ conditions, then ZPZ of this safe route is said to be positive. In other words, the requested safe route reaches a status called ZPZ positiv. The remaining procedures and statuses of a safe route are defined for monitoring these settings continuously until the safe route is properly used and released.

2.3 Interlocking System Requirements Specifications and their Problems

In the German railway domain, interlocking system requirements specifications are mostly written in German. One of the example of these specifications is [Tut06]. This specification shares similar characteristics with the other interlocking system requirements specifications. One of the characteristics is various system requirements are specified in this document (e.g. the GUI of the interlocking system). As it has been mentioned before, these characteristics might bring some problems to the system development teams who need to implement the interlocking system. Therefore, the characteristics of these system requirements and the possible problems must be analyzed based on an example. They are summarized and discussed based on the example LH-ESTW-R as follows:

- System requirements are written in natural language. Some sentences that are written in natural language are ambiguous and unclear. Terms are often not precisely defined. Ambiguous statements can be interpreted in different ways by different professions. For example,

  Example 1

No.	System Requirement
5.2.2	*Es kann angezeigt werden: Ks1 oder Ks2 und zusätzlich Zs3-Anzeige.*

Table 2.1: System requirement 5.2.2

The sentence in Table 2.1 can be interpreted in the following two ways:

"It can be indicated: Ks1 or Ks2 and in both cases an additional Zs3-Indicator" (German: Es kann angezeigt werden: Ks1 oder Ks2 und in beiden Fälle zusätzlich mit Zs3-Anzeige)

or

"It can be indicated: Ks1 or it can be indicated: Ks2 and additional Zs3-Indicator" (German: Es kann angezeigt werden: Ks1 oder es kann angezeigt werden: Ks2 und zusätzlich Zs3-Anzeige)

As it has been mentioned above, the system development teams design and develop an RIS based on the system requirements. When a member of the teams implements the requirement stated in this example, then it might cause confusion. One needs to decide the interpretation that will be taken for the implementation. The development process might become inefficient, because further discussions between team members from different disciplines will take place to consider the choices of these interpretations.

- A part of system requirements is written and defined based on complex railway concepts and scenarios. For example,

Example 2

No.	System Requirement
2.3.7	*2. Einfahrt der zu kuppelnden Züge aus entgegengesetzter Richtung* *Für beide Züge ist das Zugstraßenziel der jeweilige fiktive Zielpunkt mit D-Weg null (Gegenfahrausschluss)*

Table 2.2: System requirement 2.3.7

The system requirement in Table 2.2 consists of three complex railway concepts: the virtual route destination's point (German: fiktiver Zielpunkt), the length of the overlap is zero and locking of conflicting routes from the opposite direction (German: Gegenfahrausschluss). It is not easy to understand the relationship between these three concepts immediately. Many issues need to be considered and analyzed in order to implement this requirement. One of these issues is the role of this virtual route destination's point w.r.t. the developing requested safe route, in other words, the reservation of the route destination's point as a route or an overlap element. Since the length of the overlap is zero, one can interpret the concepts as no reservation of the point as an overlap element is needed. However, if this is implemented, then conflicting routes cannot be excluded.

This type of system requirements can be easily understood by railway engineers. However, for the other members of the system development teams, like computer scientists, it is difficult to understand and analyze the meaning of these statements and concepts. Discussion among professions from different areas become complicated.

- Some system requirements which describe the same or similar concepts are defined in different chapters, based on different views and aspects. This characteristic is further discussed based on the following examples:

Example 3

Table 2.3 shows two system requirements which define the same concept, overlapping overlaps (German: überlappende D-Wege). They are specified in different chapters. Without the system requirement 1.7, one might have difficulties to understand the system requirement 2.2.1 because the definition of positions of the point (German: Lage) and the locking of the requested overlaps (the German sentence: "...den neu hinzukommenden D-Weg nicht

No.	System Requirement
1.7	*D-Weg der Zugstraße* *- DWege können sich gleich- oder gegenläufig überlappen, falls sie sich nicht über spitz zur Durchrutschrichtung liegende, verschlossene Weichen oder Weichen mit beweglichen Herzstückspitzen ausschließen.*
2.2.1	*Überlappende D-Wege* *Im Bereich Ziel-D-Weg-Ziel dürfen ein oder mehrere D-Wege vorhanden sein, wenn die verschlossenen Weichen dieser D-Wege durch ihre Lage den neu hinzukommenden D-Weg nicht ausschließen (Überlappende D-Wege)*

Table 2.3: System requirements 1.7 and 2.2.1

ausschließen") are not specified in this requirement. These two definitions have been well-elaborated in the system requirement 1.7. For example, one of the conditions of the point is that it must not be equipped with a movable frog (German: bewegliche Herzstückspitze ausschließen). In order to realize this complex scenario, one must be familiar with the specification, such that it can be correctly analyzed and understood. Furthermore, inconsistency of these requirements cannot be checked easily. If inconsistencies do exist in these specifications, then the problem of unclear interpretation might also be caused.

- The same problems are caused when the system specifications are defined w.r.t. different views of the same concepts in a chapter. For example, checking conditions of ZPZ of the requested route are written in two different views: view of a safe route (German: ZPZ-Bedingungen für die gesamte Zugfahrstraße) and view of an infrastructure object located within the route (German: ZPZ-Bedingungen je Fahrwegelement im Fahrweg). An example is shown in Table 2.4,

Example 4

Given the situation in which a point is not equipped with a movable frog (see Figure 2.7), it is then unclear whether ZPZ of the requested overlap will be evaluated to be positive w.r.t. these two specifications. Based on the system requirement 2.2.1, ZPZ of the requested overlap $DWeg_{Z2}$ is evaluated to be positive because the direction of $DWeg_{Z2}$ is coming from the trailing point and W1 is not equipped with a movable frog. However, the system requirement 2.4.1 states that a point is not allowed to be locked or reserved for another overlap in an improper position of $DWeg_{Z2}$. In Figure 2.7, the point W1 is reserved or locked for the overlap $DWeg_{Z1}$ and it is reserved to the diverging direction, as a result, ZPZ of $DWeg_{Z2}$ is negative.

No.	System Requirement
2.2	*ZPZ-Bedingungen für die gesamte Zugfahrstraße*
2.2.1	*Überlappende D-Wege* *Im Bereich Ziel-D-Weg-Ziel dürfen ein oder mehrere D-Wege vorhanden sein, wenn die verschlossenen Weichen dieser D-Wege durch ihre Lage den neu hinzukommenden D-Weg nicht ausschließen (Überlappende D-Wege)*
2.4	*ZPZ-Bedingungen je Fahrwegelement im Durchrutschweg*
2.4.1	*ZPZ-Bedingungen für Weiche ohne bewegliche Herzstückspitzen* *a)Die Weiche darf nicht in der Nicht-Solllage gegen Umstellen gesperrt sein.* *a)Die Weiche darf nicht in der Nicht-Solllage beansprucht sein.*

Table 2.4: System requirements 2.2.1 and 2.4.1

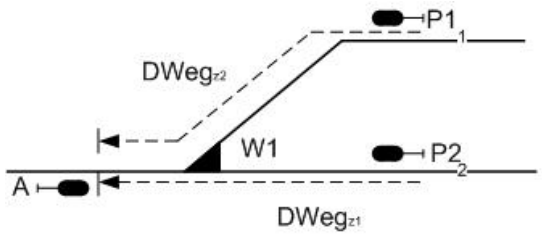


Figure 2.7: Overlapping overlaps

- The specification [Tut06] consists of different types of system requirements of an RIS, two of them are the GUI of the RIS and the interlocking logic. Some of the requirements consist of a mixture of specifications that are related to different functions of the RIS.

Example 5

No.	System Requirement
2.1.3	*Kennzeichnung der FW-EL und grünes Band* *Ist das Ergebnis der Zulassungsprüfung entsprechend den nachfolgenden Bedingungen positiv (ZPZ positiv), werden die Fahrweg elemente (FW-EL) für die Zugstraße markiert und auf der Anzeige darf das grüne Band vom Start bis zum Ziel angeschaltet werden. Belegte Gleisabschnitte oder solche mit gestörter Gleisfreimeldung bleiben rot ausgeleuchtet.*

Table 2.5: System requirements 2.1.3

With the title of the system requirement in Table 2.5, it is classified as a system requirement for the GUI of the RIS. However, it also consists of a system requirement for the interlocking logic which is the German statement "Ist das Ergebnis der Zulassungsprüfung entsprechend den nachfolgenden Bedingungen positiv (ZPZ positiv), werden die Fahrwegelemente (FW-EL) für die Zugstraße markiert." This means, if the elements within the requested safe route fulfill the corresponding ZPZ conditions, then these elements are marked as the elements of the requested safe route. It is unclear whether this marking shall be only indicated in the GUI, or it is the logical reservation of elements for the requested safe route.

Unclear definitions might cause confusions and wrong interpretations to the system development teams. Confusion can be clarified by further discussions among members from different development teams. This costs time and resources. However, a wrong interpretation might cause errors in design or implementation. In this example, if this statement was classified as a requirement for the GUI only, then a logical reservation of a route element is not made in the developed RIS. The costs of amending errors are very high when they are found later in the system development process. Furthermore, the wrong interpretation might also bring difficulties for the analysis of the correctness of the specification because some system requirements for the interlocking logic are missing for analysis and verification. To illustrate this problem, the following artificial statement is considered:

No.	System Requirement
x	*Positive Zulassungsprüfung ZPZ* *Nach positiver Zulassungsprüfung dürfen die Fahrwegelemente (FW-EL) für die Zugstraße nicht markiert werden.*

Table 2.6: An artificial system requirement

This statement is classified as system requirement for the interlocking logic. It would be clearly inconsistent to the system requirement 2.1.3. However, it might not be possible to check this type of inconsistency if the developer classified the system requirement 2.1.3 as a specification for the GUI of the RIS.

Since some specifications are mixtures of requirements from different types, members of the system development teams cannot obtain an overview over the system requirements quickly. System development teams are composed of different groups. Each group is assigned to specific implementation and design tasks. For example, a team is responsible to implement the GUI of the RIS exclusively. Based on this property of this specification, it is not easy to assign tasks to each team efficiently.

In conclusion, the specification is written in German natural language. Some system requirements that are written in natural languages might be unclear. Unclear sentences cause problems for the system development teams to understand and interpret the requirements. Further discussions must take place in order to

consider and solve the unclearnesses and possible interpretations. This increases the development costs of the system. If this type of unclearness can be avoided in this early stage of a system development process, the system development will become more cost effective and efficient. Furthermore, some requirements are specified based on complex scenarios and railway terms. Discussions among experts from different professions might become complicated. Computer scientists have difficulties in analyzing these concepts for the implementations. Finally, the correctness of the specification is difficult to be checked because it is written in a natural language and it is written in a particular structure (e.g. a requirement specification written in different chapters, aspects and views). Incorrect and complicated system specifications might lead to a wrong interpretation resulting in an error in the system software. If this error is found in the later stage of the system development process, it then becomes expensive to correct this error (see Figure 2.8). In the German railway domain, the writing style of interlocking system requirements is coherent. Therefore, the stated characteristics and problems will also appear not only in this analyzed specification, but also in other interlocking system requirements specifications. It is then important to search possible solutions to deal with these problems.

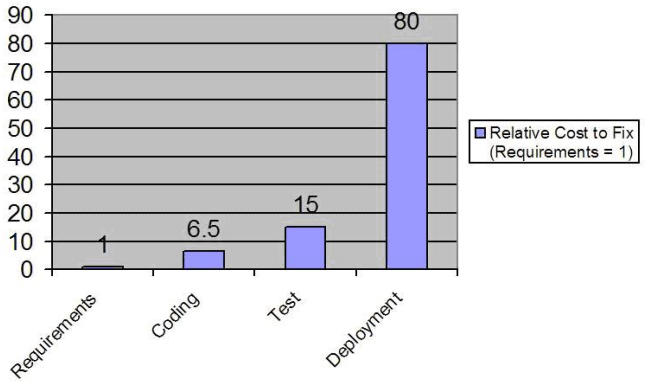


Figure 2.8: Costs for correcting an error during different phases in a system development process [KM08]

Chapter 3

Formal Methods and their Applications to the Railway Domain

The characteristics of the system requirements of regional computer-based RIS and the possible problems that they might bring to system development teams have been discussed in section 2.3. One of the possible ways in helping to lessen the effect of these problems is the application of formal methods. This is one of the goals of this research work, investigating the reasons and possibilities for applying formal methods to specify interlocking system requirements.

In this chapter, a general introduction to formal methods is given. The advantages of applying them to specify interlocking system requirements are then discussed. The last section of this chapter describes the current applications of formal methods in the railway domain and their results. Furthermore, the experience that are gained from the studies and their applicability to this research work are discussed.

3.1 Formal Methods

Formal methods provide a mathematical framework to users for specifying and analyzing system requirements [Win90, Mon03, TE04, Bow97]. Tools are implemented based on the mathematical concepts of frameworks to verify a system design against these system requirements. As a result, applying formal methods to develop systems becomes an important research topic [HB95].

In this section, a general introduction to formal methods is first given. The advantages of applying them in the system development process are then discussed.

3.1.1 Introduction to Formal Methods

System requirements are said to be formally specified if they are expressed by applying mathematical concepts and mathematical logic. Formal methods provide a framework to specify system requirements formally. The first component of a

formal method is a formal specification language [AP98]. The language comprises syntax (notations) and semantics (meaning of the notations). The semantic meaning of every notation is mathematically and clearly defined. As a result, one can formalize the same concept with equivalent notations and if the formalizations are correct, there will be only one meaning of those formalizations. Propositional logic (see Section 5.1) is one of these mathematical specification languages. For example,

Example 6

No.	System Requirement	logical meaning
2.3.1	*ZPZ-Bedingungen für Weiche* *a)Die Weiche darf nicht in Nicht-Solllage verschlossen sein.*	$\neg(\neg Solllage \wedge EL(v))$

Table 3.1: The system requirement 2.3.1(a)

Solllage: The object is in the proper position for the requested safe route.
Das Element liegt in der Solllage für die einzustellende Zugstraße.

$EL(v)$: The object is locked as a route, an overlap or a flank protection Element of other routes.
Das Element ist als Fahrwegelement oder DWeg-Element oder Flankenschutzelement für eine andere Zugstraße verschlossen.

$(Solllage \vee \neg EL(v))$ means, if the point is in a proper position, it is not important whether it is locked or not, however if it is not in the proper position, then it should not be locked. One can also formalize the sentence equivalently as $(EL(v) \rightarrow Solllage)$. This sentence can be read as "if the point is locked, then it is also in the proper position for the requested safe route.". Although the concept is represented by different notations, the semantic meaning has not changed and there is only one way to interpret the concept. This equivalent relationship can be found by the syntactical part of the language. For example,

$$\begin{aligned} Requirement_{2.3.1} &\equiv \neg(\neg Solllage \wedge EL(v)) \\ &\equiv Solllage \vee \neg EL(v) \\ &\equiv EL(v) \rightarrow Solllage \end{aligned}$$

The second component of formal methods is analysis and verification of well-formalized specifications. If the methods for verifying and analyzing specifications are well-defined, computer programs can be implemented based on the concepts. The tasks can be carried out automatically. For example, a well-formalized specification or design can be verified against the expected behavior of the system [Som04, Koo99]. One of the formal verification tools is a model checker. A

model checker implements the concepts of the formal method called model checking [CGP99]. One can formalize a design of an interlocking system as a transition system and the expected behavior of the system as checking conditions, for example, safety requirements (German: Sicherheitsregeln) [HK06]. The model checker verifies the transition system against checking conditions. It produces a counterexample if the transition system does not conform with the condition.

3.1.2 Advantages of Using Formal Methods

The above discussed components of formal methods can bring advantages to users in specifying system requirements, like [Tut06] as follows:

- Reduction of ambiguity of specifications: The well-defined formal specification language can help to avoid an unclearness in a specification because each requirement must be specified as a precise mathematical statement. In Example 1 on page 10, there exists two ways to interpret the sentence in Table 2.1. These two interpretations can be defined by propositional logic as follows:

  Example 7

No.	System Requirement	logical meaning
5.2.2	*Es kann angezeigt werden: Ks1 oder Ks2 und zusätzlich Zs3-Anzeige.*	First interpretation: $(KS1 \vee KS2) \wedge Zs3$
5.2.2	*Es kann angezeigt werden: Ks1 oder Ks2 und zusätzlich Zs3-Anzeige.*	Second interpretation: $KS1 \vee (KS2 \wedge Zs3)$

Table 3.2: The system requirement 5.2.2 and logical meanings

$KS1$: The object indicates Ks1.
Das Element zeigt Ks1 an.
$KS2$: The object indicates Ks2.
Das Element zeigt Ks2 an.
$Zs3$: The object indicates Zs3.
Das Element zeigt Zs3 an.

By applying formal methods in the development process, system requirements need to be specified based on the formal specification language of the corresponding formal methods, each requirement becomes an explicit mathematical statement, like in Example 7, the specifier can either define the corresponding requirement as $(KS1 \vee KS2) \wedge Zs3$ or $KS1 \vee (KS2 \wedge Zs3)$. Furthermore, a single interpretation is deduced from the statement and misleading interpretations can be avoided (see Example 6 on page 17). When the specification is unambiguous, it is easier for professions from different areas to discuss the specification or understand the system and the discussion

becomes efficient because terms are well defined and equivalent concepts can be deduced by the mathematical rules. In the case of computer-based interlocking systems, if the specification is expressed by mathematical concepts or logics, computer scientists who do not have sufficient knowledge in railway systems can relatively easy understand the specification. This can increase the efficiency of the software development process and the correctness of software systems.

- Increasing the understandability of specifications: It is sometimes difficult to understand and analyze the meaning of a requirement that is written in natural language (see Examples 2 on page 11 and 3 on page 11). With the help of mathematical concepts, one can capture the meaning and concept of a statement efficiently. The requirement 1.7 that is specified in Table 3.3 can be specified as follows:

Example 8

No.	System Requirement	logical meaning
1.7	*D-Weg der Zugstraße* *- DWege können sich gleich- oder gegenläufig überlappen, falls sie sich nicht über spitz zur Durchrutschrichtung liegende, verschlossene Weichen oder Weichen mit beweglichen Herzstückspitzen ausschließen.*	$\neg(\neg Solllage \wedge Spitz \wedge DWEL(v)) \wedge \neg(\neg Solllage \wedge BHSS \wedge DWEL(v)))$

Table 3.3: The system requirement 1.7 and the logical meaning

$DWEL(v)$:	The object is locked as an overlap element of other routes. Als DWeg-Element (DW$\geq$0) für eine oder mehrere Zugstraßen verschlossen.
$Spitz$:	The requested overlap of the safe route is a facing point movement. Der einzustellende DWeg läuft über eine spitz zur Durchrutschrichtung liegende Weiche.
$BHSS$:	The object is equipped with a movable frog. Das Element ist mit beweglicher Herzstückspitze ausgerüstet.

$$\begin{aligned} Requirement_{1.7} &\equiv \neg(\neg Solllage \wedge Spitz \wedge DWEL(v)) \wedge \neg(\neg Solllage \\ &\quad \wedge BHSS \wedge DWEL(v)) \\ &\equiv \neg DWEL(v) \vee Solllage \vee (\neg Spitz \wedge \neg BHSS)) \end{aligned}$$

The concept of overlapping overlaps has been illustrated mathematically in this example. The propositional formula $\neg DWEL(v) \vee Solllage \vee (\neg Spitz \wedge$

$\neg BHSS$)) expresses the meaning of this concept[1]. One can use this formula to check whether the requested overlap is considered as an admitted overlap. The cases are summarized as follow:

1. The point is not locked for another route as an overlap element.
2. The point is in a proper position for the requested overlap.
3. If the point is not in a proper position and locked for another safe route as an overlap element, then the requested overlap must be a facing point movement and the point must not be equipped with a movable frog.

The concept that is written in natural language in Table 3.3 is now easier to understand. Furthermore, a given situation can be analyzed with the help of this logical formula (see Figure 3.1 and 3.2). The number of possible situations that are described by the concept can also be generated.

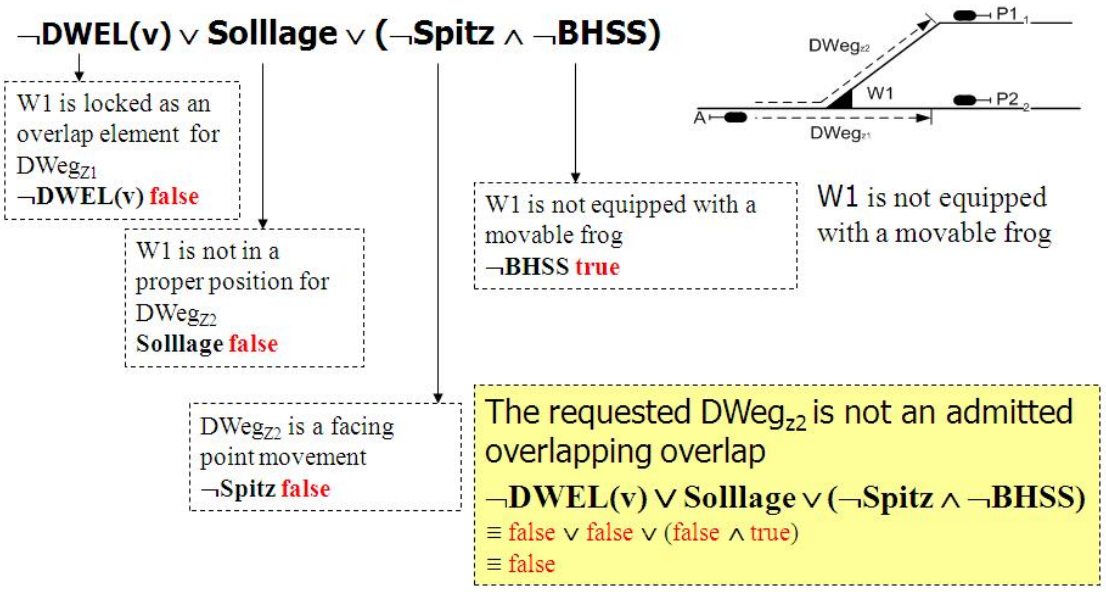


Figure 3.1: Analysis of a situation

[1]Requirement 1.7 is a general specification, other situations w.r.t. this concept will be fully elaborated in another part (e.g. ZPZ conditions) of the interlocking system requirement specification.

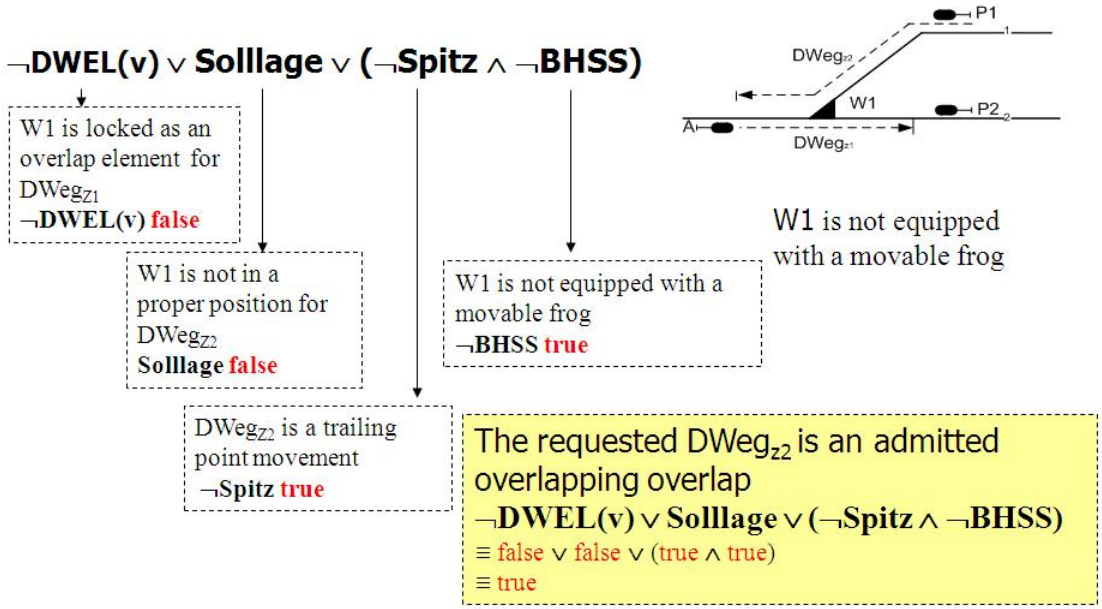


Figure 3.2: Admitted overlapping overlaps

- Checking the correctness of specifications: In formal methods, this is achieved by reasoning of the formal specification (see Section 5.1.3). For example, Figure 3.3 shows a system development process called V-model. In the stage of system requirements, these requirements are specified with a set of logical statements. When it is noticed that there exists a statement which is true and false at the same time in the set, then inconsistency is said to be found. Furthermore, equivalent concepts can be deduced by applying the syntax rules of the mathematical language. A concept or system requirement in different views or aspects can be first specified as two formulas. If these two formulas are not equivalent, then an unclearness in the specification exists. For instance, the type of unclearness that has been stated in Example 4. Formal methods support proving the conformity of the specification to the expected behavior of the system. If the specification does not fulfill the expected behavior, it means that the system has not been specified completely or correctly. For example, the result produced during the phase of system requirements can be verified against the output from the phase of user requirements, it can check whether the developed system requirements satisfy the requirements from the users. In other words, if the system requirements specification, like [Tut06], can be verified against the safety requirements, then the systems that are developed based on this specification also satisfy the safety requirements.

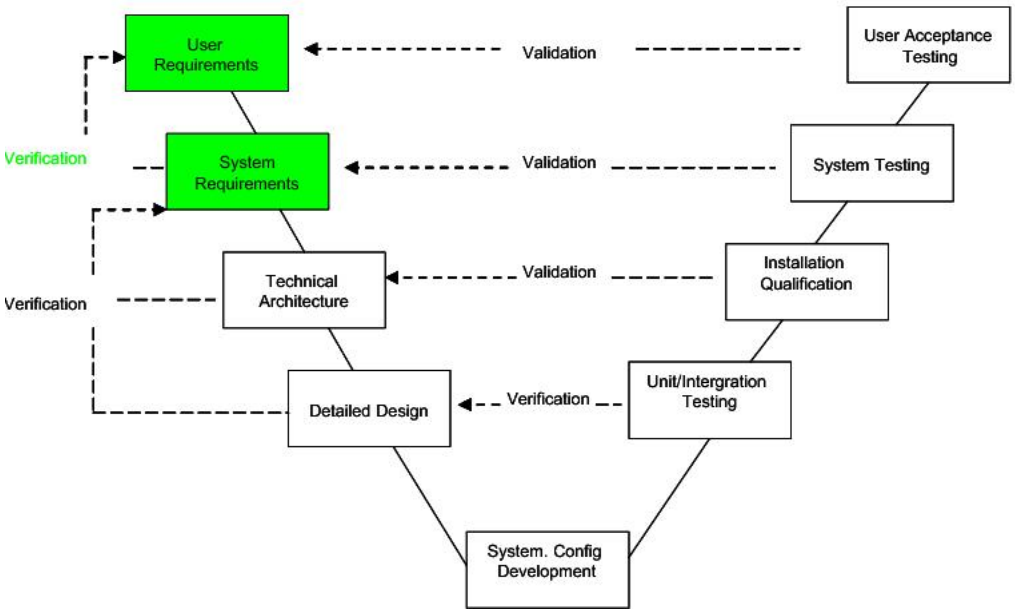


Figure 3.3: System development process, V-model

3.2 Applications of Formal Methods to the Railway Domain

Because of the mentioned advantages, the importance of using formal methods to specify and verify systems has been addressed in different fields, especially those which develop safety critical systems [BC00]. The CENELEC railway application standards [CEN99, CEN00b, CEN00a] are focused on the standardization of different European railway principles. The standard of developing railway software has also been discussed and formal methods are suggested to be used in order to increase the safety integrity levels of the softwares. As a result, investigations of applying formal methods to the railway domain have become an active research area. There are various research topics: defining railway specific languages by using formal methods, formal verification of and automatic formal development of railway control systems.

In this section, formal methods that have been investigated in the stated topics are briefly introduced. Some of the corresponding investigations and their results are discussed.

3.2.1 Set-theoretic Specifications

Z [Spi88] (Z notation), VDM [Jon90] (Vienna Development Method) and B-Method [Sch01] are set-theoretic specifications. The mathematical concept of Z, VDM and B-Method is set theory [Hal60]. In these formal methods, a system or an object is composed of states which are defined as sets. The changes of the states are specified as the operations over the defined sets. These operations are described as

pre- and post- conditions of sets and set-theoretic operations are used to describe these conditions. In these three methods, the style of the specification languages are slightly different from each other. However, the mathematical theory behind is the same. As it has been mentioned above, when the specification language is well-defined, tools can be developed to check and verify specified models. Tools for Z, VDM and B-Method have been developed [Pro07, Ade08, Cor08]. The verification of the model against the expected behavior is achieved by generating proof obligations in these tools. The expected behavior is specified as invariants in these three methods. An invariant is a condition which must always hold true in the system or object.

One of the applications of set-theoretic specifications in railway domain is presented in [BS98]. In this project, a domain specification language for railway systems is developed. Defining a domain specification language is one way to achieve the standardization of systems [EAF99] which is addressed in the CENELEC railway application standard. A domain specification language is used to define the meaning of every concept of a domain. These concepts are modeled by the chosen modeling language (e.g. Z). One of the domain concepts in railway systems is the meaning of a point. Once the domain specification language has been developed, a railway system (e.g. an RIS) can be described by all these defined concepts. In this project, concepts of railway systems, like routes, trains and functions of railway systems, are defined by using RSL (Rigorous Approach to Industrial Software Engineering Specification Language) [Gro95]. This language is developed based on a number of formal languages. One of them is VDM. The specified railway concepts are represented by the notations of set theory clearly. In order to use and understand the defined domain language, one needs to understand the mathematical concepts behind RSL, like set theory. Other representation forms of the domain language are not provided. Therefore, the use of the language does not gain the popularity among railway engineers.

In [Han98], a design model of an RIS is formalized by using VDM abstractly. The system is defined by using states' changes of the tracks, points and signals in the VDM specification language. The model of the interlocking system is specified based on an informal system requirements specification. The possibility of checking the correctness of these system requirements and the methods of transforming these system requirements to the defined mathematical model have not been discussed in this work. Nevertheless, in order to understand the specified model and the invariants for the verification, one needs to have the mathematical background in set theory.

Cooperations between commercial companies and the French railway company have been established for applying B-Method to verify and develop French interlocking systems [LSP07]. One of them has been presented in the papers [DM95, Mej98]. In this project, an existing system that operates the Paris fast subway RPE line A is specified and verified by B-Method. The existing informal design documents are modeled as mathematical models with the notations of B-Method by a team. This team is composed of formal experts and development engineers. The development engineers are trained for six or more months in an

intensive course of B-Method before the start of the project. Applying B-Method can increase the safety integrity of the system and the system requirement specifications become clear in this project. However, in [DM95], it has been stated that the application of B-Method has not been optimized because of the difficulties of engineers to apply mathematical concepts. Extensive guidance to the usage of methods is needed in order to improve the quality of the usage of B-Method in the project.

In these mentioned projects, the mathematic system models are checked against the expected behavior of the systems. Therefore, the safety integrity of these systems is increased. Defined domain concepts are clear and unambiguous. However, it has also been shown that in order to use Z, VDM and B-Method, the users need to be familiar with set theory. Railway engineers might have less training in this respect. As a result, system models are mostly defined by formal experts and the participation of railway engineers in the system development process is relatively small because they might have difficulties in understanding and analyzing the meaning of those specified set-theoretic models.

3.2.2 Abstract Algebraic Specifications

Another formal method that has been investigated in the railway domain is abstract algebraic specification. Maude [Mes06] is an example of abstract algebraic specifications. The mathematical concept behind this formal method is abstract data types [Mon03]. A system or an object is modeled as an abstract data type. A sort and signature are defined for the data type. A sort is the name of the system or object, while a signature is used to define the operations on the system or object, respectively. The interpretation of an operation can be expressed as a predicate axiom or an algebraic equation. If the interpretation is written as an equation, then this abstract data type is called an algebraic abstract data type. And the formal methods in which systems and objects are modeled by algebraic abstract data types is called the abstract algebraic specification. If safety requirements and expected behavior of the system or object are specified by algebraic abstract data types, the object can be verified against safety requirements by equational reasoning of the data types. An extension of algebraic specification is the process algebraic specification [Bae05], like CSP (Communicating Sequential Processes) and μCRL (micro Common Representation Language) [Gro97]. Process algebraic specifications are used to model distributed systems. These systems are composed of a number of processes. Mechanisms for defining communications between processes and the concurrency of a system are defined in this specification. Tools has been developed to support specification and verification of systems in abstract algebraic specifications [CEW93].

In [HP00], a formal development process for a distributed railway control system is introduced. In this distributed railway system, no railway signals are used. The formal method that is used to support the development process is process algebraic specification, CSP. The main concept of this case study is to develop the formal specification of the system and safety requirements (e.g. no collision) step-

wise and ensure the correctness of the formal specification in each step. Therefore, in each step, the specification is verified against the safety requirements by equational reasoning. They are both refined when the specification fulfills the safety requirements. Railway engineers participate only in the beginning of the whole development process to describe the system properties. No transformation of the specification and safety requirements from the algebraic specifications to another semantic form has been mentioned. In order to develop a distributed railway control system with the concepts of this work, one needs to be familiar with abstract algebraic specifications.

In [GVVK95], an developed interlocking system is verified by applying the process algebraic specification. This interlocking system was implemented by VPI (Vital Processor Interlocking). The system is expressed in the specification language called μCRL as a model and safety requirements are specified in modal logic [BdRV01] . μCRL is a type of process algebraic language. A specifier uses abstract data types to define the system model in μCRL. Modal logic is a type of temporal logic which can be used to specify the checking conditions of a transition system (see Section 3.2.3). A verification tool is used to check whether the model fulfills the specified requirements. If the model satisfies the requirements, the truth value true is returned, otherwise false. In this case study, it ensures the modeled interlocking system satisfies the defined checking conditions. However, the correctness of the defined checking conditions depends on the understanding of formal experts w.r.t. the railway safety requirements. No specific methods have been developed to check the correctness of the requirements and help railway engineers to be involved in the process.

3.2.3 Behavior Specifications

Transition systems, finite state machines and temporal logics [HR04] are mathematical concepts that are commonly used for behavior specifications. In a transition system or a finite state machine, the system or object is composed of states. They change from one state to another when the corresponding defined condition is fulfilled, this condition is called a triggering condition. Statecharts [Har87] and UML state machines [omg05] are transition diagrams that represent a finite state machine graphically. Temporal logic is used to describe general and abstract conditions that the system or object need to satisfy over the evolvement of the system. The verification technique model checking [CGP99] combines these two concepts together. In model checking, the system is first modeled as a transition system and the expected behavior of the system is defined in a temporal logic. A model checker will then verify whether the general condition is satisfied in all the possible states of the system. An example of a model checker is SMV (Statistical Model Validation) [CGP99].

Statecharts have been chosen as one of the modeling languages to express the functional requirements of interlocking systems in the Europen project Euro-Interlocking [KE03] and INESS [INE09]. Although checking the correctness of the functional models has not been addressed in both projects, a number of research

projects were done to demonstrate the feasibility of verifiying behavior specifications in the railway domain. In [PGHD04], the concept of applying model checking to an automatic railway control system development has been investigated. In this case study, a tram control system is developed by the proposed concepts in the paper [HP00]. The generic behavior of train movements and generic control rules of the control system have been first defined as transition systems [HP02]. Then, railway engineers provide a route table, signal setting table and the locking table to formal experts. Based on this configuration data, an executable control system which develops safe routes will be produced. In order to verify this system, safety requirements are defined informally first and they are transformed into temporal logic. The transition system is then verified by a model checker. If the system satisfies safety requirements, it will be transformed into executable code for implementation. In this project, specification of the logic of the generic system (e.g. the triggering conditions of the generic system) is done by formal experts. It provides no chance for railway engineers to analyze the defined transition models and temporal logical formulas. Therefore, without the mathematical background, railway engineers obtain less chance to specify the generic behavior of the train movements and generic control rules.

Other research activities applying model checking to the development and verification of interlocking systems have been carried out [CGM+98, Win02]. These projects share the same characteristics as the project in [PGHD04]. Applying model checking in the railway system development process provides a chance to check the correctness of the system model against safety requirements of the system. However, the specification of the transition system and safety requirements is achieved by formal experts instead of railway engineers. Understanding and interpretation of system properties for formal experts and railway engineers might vary. One can only assume the defined model captures the whole system behavior and temporal checking conditions are correctly translated from the safety requirements. Since the transition system grows more complex with capturing the behavior of the system in more detail, one will have difficulties to understand the model. No other form of representing the semantics of the mathematical models has been investigated in order to help users to understand the system model.

3.2.4 Experience to Motivation

In conclusion, the discussed projects in this section concentrate on applying formal methods to the whole system development process (see Figure 3.3). The importance of checking the correctness of informal system specifications with mathematical concepts is not addressed. In other words, unclearness (see Example 4 on page 12) that exist in an informal system requirements specification cannot be found based on the concepts of the mentioned projects. Before applying those formal methods that have been described above, formal experts need to understand the domain concepts and the informal system requirements specification in oder to model the system. Railway engineers describe the system properties and domain concepts to formal experts and assist them to understand the informal

specification. Formal experts are then responsible to write down the requirements in formulas and apply the corresponding formal specification language to complete the models. If the unclearness in the system requirements specification has not been found before, formal experts and railway engineers might overlook the unclearness during their discussion about the informal specification. The defined model might be wrong. This can also be applied to the specification of safety requirements as mathematical checking conditions. Therefore, the correctness of this informal system requirements specification must be checked.

The mathematical system is first modeled based on the informal system requirements specification. Since the modeled system is described by mathematical concepts, the specifications become now clear and well-defined. In other words, computer scientists and railway engineers can understand the informal specification by using this well-defined system model. As it has been mentioned before, the well-defined system model is developed by formal experts. It is important for railway engineers to examine this defined model because understanding and interpretation of professions from different disciplines might vary, especially in the cases of modeling complex railway concepts and scenarios (see Example 2 in Section 2.3). However, the mathematical model might become complex and one needs to be familiar with the corresponding mathematical concepts in order to examine and understand the model. In the mentioned projects, railway engineers are not provided with any assistance to understand and analyze the models or formulas. They play a passive role in the whole formal development process. As a result, applications of formal methods or mathematical concepts among engineers or industries are not popular because the involvements of railway engineers in the specification and verification process is not sufficient enough [Kni98].

This experience plays an important role in this research work. Based on the above analysis, the focus of this research work is put on applying formal methods in the phase of system requirements specification of the system development process. Based on the characteristics of interlocking system requirements, a formal framework should be developed. This framework provides the suitable mathematical concepts or specification languages to specify the interlocking logic formally. In the mentioned projects, system models are difficult to be interpreted because of the complexity of the specification languages. The chosen specification language must be simple and it must be sufficient to specify requirements. Furthermore, it is important to find out the incorrectness that might exist in informal system requirements specifications. In the discussed projects, an important experience is gained. In order to increase the involvements of railway engineers and their confidence in applying mathematical specification languages at this stage of the system development process, transformations of the formal specification into a semantically equivalent (see Section 5.1.3 for the meaning of semantic equivalent) form must be provided. Secondly, railway engineers must be able to specify and check the correctness of the specified interlocking logic in the framework. In other words, this formal framework must be engineering-oriented.

Chapter 4

The Engineering-Oriented Formal Framework – Object-Based Lastenheft

The characteristics of interlocking system requirements, the advantages of using formal methods and the application of formal methods to the railway domains have been discussed in the previous chapters. In this work, concepts are developed to handle the discussed problems and are conceptualized into a formal framework called Object-Based Lastenheft (OBLH). This framework is composed of a specification language, methods and an implemented tool. These components support different professionals, especially railway engineers, to formally define and examine interlocking system requirements specifications. Furthermore, the developed concepts are mathematically well-defined.

In this chapter, an overview on OBLH is first given in Section 4.1. The reasons for developing the concepts and the advantages of applying them are elaborated in detail in the remaining sections of this chapter.

4.1 An Overview on Object-Based Lastenheft

The main goal of this research work is to develop an engineering-oriented formal framework that supports specification, analysis and verification of interlocking system requirements specifications. Based on the described properties of an interlocking system requirements specification (see Section 2.3) and others experiences of applying formal methods to the railway domain (see Section 3.2.4), a formal framework called Object-Based Lastenheft (OBLH) is developed to achieve this goal [HGE08]. The word "Object-Based" means that every infrastructure element (e.g. point) that belongs to a safe route is considered as an object and the object consists of attributes (e.g. $BHSS$: The object is equipped with a movable frog and in German: Das Element ist mit beweglicher Herzstückspitze ausgerüstet.). While the word "Lastenheft" stands for the ideas that each system requirement of the specification is expressed by the objects' attributes with the appropriate mathematical specification language.

In this framework, different types of system requirements (e.g. system requirements of the GUI) that have been expressed in [Tut06] are defined and the system requirements can then be classified into the corresponding group based on the definitions (see Section 4.2.1). An interlocking system requirements specification becomes then well-structured.

Based on the study of the characteristics of the interlocking system requirements specification, the interlocking logic is generally defined by two types of conditions in OBLH (see Section 4.2.2). They are static conditions and dynamic conditions. The static conditions are used to describe the checking conditions or constraints that the route, overlap and flank protection elements of a safe route need to fulfill in the procedures of a safe route development and statuses of the safe route. The dynamic conditions describe these procedures and statuses and also their relationships. The focus of this research work is first put on supporting specification and analysis of static conditions. The suitable mathematical specification language to specify static conditions is propositional logic (see Section 5.1). With the well-defined semantics and syntax of propositional logic, the checking conditions and complex railway scenarios can be clearly and formally defined.

As it has been mentioned in Section 3.2.4, the specified formal model must be understood by railway engineers and computer scientist easily. To express the meaning or semantics of each propositional formula, truth tables [Hod77, HH02] (see Section 5.1.2) and decision tables [Mon74, Har06, Hur82, Str77, Jüt89] (see Section 5.2) are used. One of the reasons to use decision tables to represent the semantics of propositional formulas is that decision tables are well known for their simplicity to represent decision rules [GP95, Con79, VD94a, Van05, Lau65]. In addition, propositional formulas and decision tables are semantically equivalent [ZB99] (see Section 5.3). This transformation provides also a chance for the users to check and inspect the correctness of the formally specified requirements (see Section 4.3.1).

Decision tables are not only used to represent the meaning of propositional formulas in OBLH. They can also be used to specify static conditions of the interlocking logic and the safety requirements for checking the correctness of the system requirements. The specified decision table must be well-formed. In order to support the users in specifying the interlocking logic in a well-formed decision table, the completeness w.r.t. the specified conditions of a decision table and the consistency of rules of a decision table can be checked (see Section 4.3.3).

Some decision tables are composed of many rules if the defined system requirements are complex. To support analysis of a decision table, one can specify a situation that needs to be evaluated w.r.t. the decision table. The evaluation result is then generated based on the defined rules in the decision table (see Section 4.3.2). For example, if the railway concept overlapping overlaps is defined in a decision table, with a given situation (see Figure 3.1 and Figure 3.2 on page 20), the evaluation result is generated.

Once the system requirements are syntactically well-defined based on the formal specification language, finding inconsistencies, unclearness and checking correctness of the system requirements can be achieved by the defined mathematical

concepts of the specification language (see Section 4.3.4). OBLH methods (methods 1 - 4) that are based on concepts of propositional logic and the properties of interlocking system requirements specifications have been developed. Method 1 is used to detect the inconsistency between two system requirements by comparing their evaluations. Method 4 is used to check the correctness of system requirements by comparing the system requirements against the expected behavior of the system or safety requirements. Method 2 and Method 3 are applied to specify consistent and formal system requirements from the natural language written requirements. In these two methods, the consistency of static conditions is achieved by applying Method 1. All the OBLH methods are executed by using decision tables. With the usage of decision tables, railway engineers and computer scientists can participate in the process of checking correctness and finding inconsistencies of the system requirements.

As it has been mentioned in Section 3.1.1, mathematically well-defined concepts and methods can be implemented as a program, such that the corresponding mathematical operations are carried out by the program automatically. The stated concepts and methods in OBLH are mathematically specified, therefore, a tool has been implemented to support users in applying these methods and concepts. A user guide of the OBLH tool can be found in Appendix E. The functions of this tool are summarized as follows:

- Transformation of propositional formulas to decision tables or truth tables.
- Specification of the interlocking logic as well-formed propositional formulas (see Section 5.1.1 for the meaning of well-formed propositional formulas).
- Specification of the interlocking logic in well-formed decision tables. Completeness w.r.t. the given conditions of a decision table and consistency amongst rules of a decision table are examined.
- Comparison of system requirements (in form of a decision table or a propositional formula) with a given situation.
- Supporting specification, verification and analysis of interlocking system requirements based on the defined steps in OBLH methods.

The above stated concepts are summarized as an activity diagram in Figure 4.1 on page 32. The process in this activity diagram (the left-most activity partition) indicates the usage of OBLH concepts to develop formal and correct system requirements from a natural language written system requirements specification. This process can also be used to develop new interlocking system requirements specifications formally. The OBLH concepts that support this process are drawn in four other activity partitions, for example, during the process, if system requirements are defined in a well-formed propositional formula, this formula can be transformed into a decision table for a better analysis. This is achieved by the activity illustrated in the second-left activity partition.

The mentioned concepts and methods will be further elaborated with given examples in detail in the following sections. The mathematical approaches of OBLH methods and the mentioned concepts (e.g. the transformation of a propositional formula to a decision table), the underlying technique that supports the implementation of concepts are discussed in Chapter 5. The introduction to propositional logic and decision tables is also given in the same chapter.

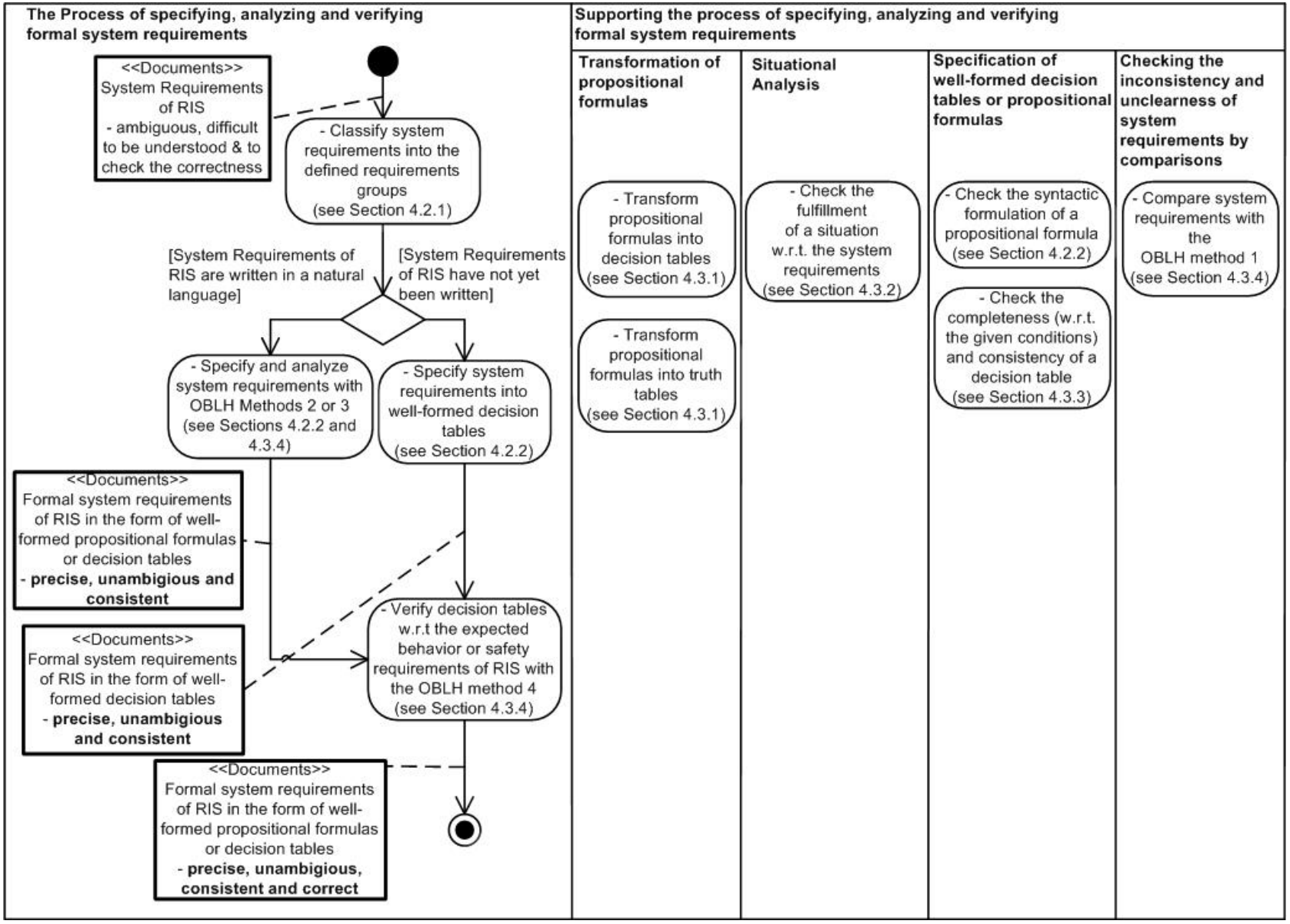


Figure 4.1: Applications of OBLH

4.2 Specification in Object-Based Lastenheft

Specification in OBLH provides mechanisms to users to structure interlocking system requirements specifications and to express these requirements explicitly and formally.

In Section 4.2.1, the definitions of classification groups and the possible advantages of the classification are discussed. The specification of interlocking logic in OBLH is discussed in Section 4.2.2 and the concepts of static conditions and dynamic conditions are elaborated in more detail.

4.2.1 Classification of System Requirements

In interlocking system requirement specifications, some system requirements are composed of specifications for different functionalities of an RIS, for example, a system requirement could consist of specifications for the GUI and the interlocking logic. However, such a composition is not clearly stated in this system requirement. The problems have been discussed and illustrated by an Example (see Example 5 on page 13) in Section 2.3. In order to handle these problems, the possible groups of interlocking system requirements are defined. System requirements can then be classified into the corresponding groups in OBLH.

The groups of interlocking system requirements are listed as follows:

- Planning and projection: It defines the properties and attributes of safe routes that the development teams need to consider and specify in the RIS during the planning and projection of the system. The functionalities of the interlocking logic are defined based on these properties and attributes. An example of this type of system requirement is shown in Table 4.1.

No.	System Requirement
2.1.2	*Einzurichtende Merkmale für Zugstraßen* *Folgende Merkmale der Zugstraße müssen durch Planung und Projek–tierung eingerichtet sein, damit die Zulassungsprüfung für diese Zug–straße positiv sein kann:* *– Zugstraßenstart* *– Zugstraßenziel* *– Durchrutschwegziel* *– Fahrwegweichen* *– Weichen des Durchrutschweges*

Table 4.1: A system requirement for planning and projection

- GUI: It describes the graphical indication of a safe route and the corresponding infrastructure elements on the GUI based on the statuses of the safe route and the procedures of the safe route development. For example, Table 2.5 on page 13 states a requirement of this type. The requirement says if the requested safe route fulfills all ZPZ conditions, then the band between the start and the destination of this safe route must be shown in green color.

- Interlocking logic: It expresses the logic of the RIS that the system development teams need to implement in the RIS, such that safe routes are correctly developed. System requirements of this type describe the procedures of a safe route development, the statuses of a safe route, as well as the corresponding checking conditions within these procedures and statuses. The interlocking logic can be classified into two types: static conditions (see Table 4.3) and dynamic conditions (see Table 4.2). They are further discussed in the following section.

No.	System Requirement
2.1.5	*Negative Zulassungsprüfung ZPZ* *Bei negativer ZPZ wird die Eingabe des Bedieners abgewiesen.*

Table 4.2: A system requirement for the interlocking logic, dynamic condition

With the classified system requirements, each development team can concentrate on its implementation based on the clearly defined specification. Wrong interpretations and confusion can be avoided. The system development becomes cost-effective by reducing the number of stated errors as in Example 5 on page 13 and discussions among development teams. Furthermore, the quality of verification and the analysis of the correctness of the system requirements specification are improved because of the completeness of the specification for comparisons.

4.2.2 Specification of Interlocking Logic

Among the groups of system requirements, the focus of this work has been put on specifying the interlocking logic. The classifications of this logic are discussed in Section 4.2.2.1 and Section 4.2.2.2, respectively.

4.2.2.1 Static Conditions

In OBLH, the defined procedures of a safe route development and statuses of a safe route are both described as states of a safe route. The checking conditions that an RIS needs to check within each state of the safe route are called static conditions. Static conditions summarize all checking conditions that are specified from different views or aspects (e.g. view of a safe route) and are defined by the attributes of the infrastructure objects. Each of the chapters of the interlocking system requirements specification defines such checking conditions. For example, Chapter 2 of [Tut06] defines the checking conditions for ZPZ, while Chapter 3 of [Tut06] defines the checking conditions for moving points. As it has been mentioned before, these checking conditions are specified as statements or exceptionally allowed scenarios in natural language. The statements can be related to safe routes as well as route, overlap and flank protection elements of safe routes. For example, system requirements for train couplings (German: Vereinigung von Zügen) (see Table 2.2 on page 11), are described as scenarios.

In OBLH, propositional logic is used to specify static conditions. This logic supports definitions of declarative sentences and logical reasoning. Interlocking system requirements specifications, like [Tut06], are composed of declarative system requirements. Propositional logic is, therefore, well-suited to specify static conditions. In the following example, a checking condition for moving points is specified in propositional logic:

No.	System requirement	Propositional logic
3.2	*Umstellbedingung für Weiche oder Kreuzung im Fahrweg, D-Weg und im Flankenschutzraum (a) Weichen und Kreuzungen im Fahrweg und D-Weg sowie Flankenschutzweichen müssen frei sein und dürfen nicht in der Nicht-Solllage bzw. in der Nicht-Schutzlage verschlossen oder gegen Umstellen gesperrt sein.*	$(Frei \wedge ($ $\neg(\neg Solllage \wedge$ $(EL(v) \wedge Usp))))$

Table 4.3: The specification of system requirement 3.2(a)

$Frei$: The object of the requested safe route is free.
Das für die Fahrstraße benötigte Element ist frei.
Usp: The object is a manually locking point.
Das Element ist gegen Umstellen gesperrt.

Before a point can be set to the requested position by an RIS, it must fulfill the checking conditions that are defined in Chapter 3 of [Tut06]. One of these checking conditions is specified in propositional logic in this example. This formula $(Frei \wedge (\neg(\neg Solllage \wedge (EL(v) \wedge Usp)))) \equiv (Frei \wedge (Solllage \vee \neg(EL(v) \wedge Usp)))$ means, a point can only be moved in the following two cases:

1. It is not occupied and is set in a proper position for the requested safe route.
2. It is not occupied and if it is not in a proper position for the requested safe route, then it must not be locked for another safe route and not a manually locking point.

Through the usage of propositional logic, these checking conditions are now precisely stated and no ambiguity exists. There exists only one possible way of interpreting this formula. Analysis of this formula can be carried out based on the concepts of propositional logic (see Section 5.1). Furthermore, checking conditions are easier to be understood because the logical structure of system requirements is clearly elaborated without the usage of natural language. These advantages apply also to the case of modeling checking conditions that are expressed as complex railway scenarios.

The usage of propositional logic to specify static conditions of the interlocking logic has been implemented in the OBLH tool. Users can specify the static conditions as propositional formulas. To assist the formalization of propositional

formulas, formulas that are not formalized based on the syntactic rules of propositional logic (see Section 5.1.1) will be indicated in the tool. In Figure 4.2, the input formula is $\wedge Frei$ which is not well-formed [1] because $\wedge$ is defined to be used as a connective between two propositions.

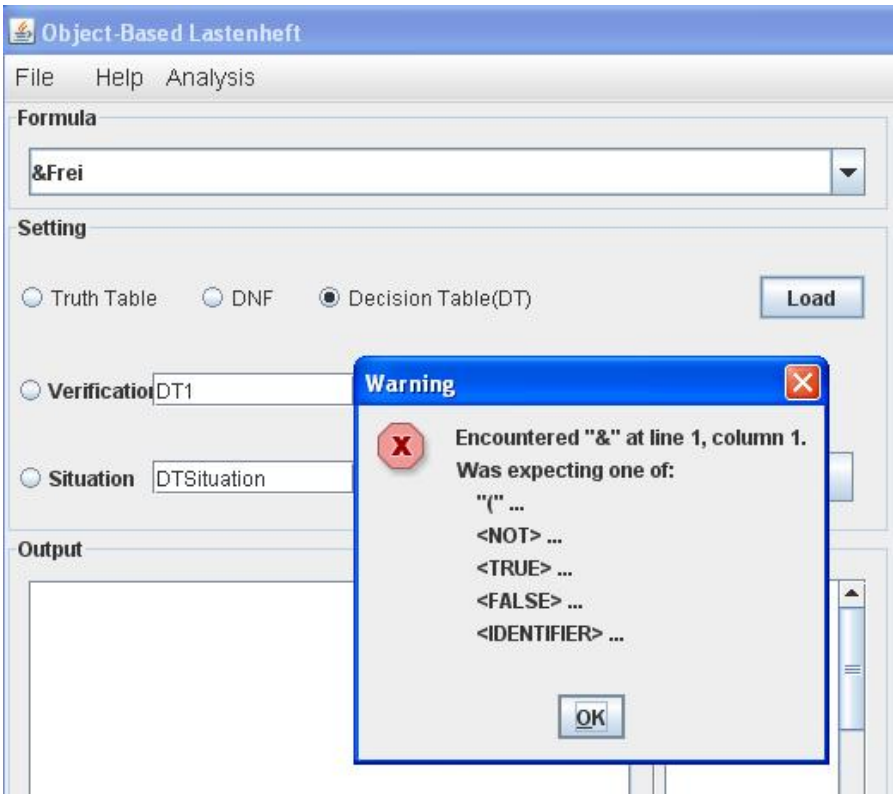


Figure 4.2: Not well-formed propositional formula

Another form of defining a static condition in the OBLH tool are decision tables (see Section 5.2). Static conditions are then specified by defining the conditions, the rules and the corresponding action of each rule. The conditions must be specified to the values `Y`, `N` or –. The symbol '–' means '*Don't-care*', while the actions must be evaluated to `Y` or `N`.

In the static condition $(Frei \wedge (Solllage \vee \neg(EL(v) \wedge Usp)))$, the attributes $Frei$, $Solllage$, $EL(v)$ and Usp are the conditions of the decision table. The rules describe situations that must be considered under these system requirements. An example of such a situation is that a point which belongs to the route of the requested safe route is free and it is not in the proper position for the safe route. This point is not locked for another safe route and not a manually locking point. This situation is represented by $R4$ in the decision table in Figure 4.3. The action of each rule is the evaluation of the situation. If the point satisfies this described situation, then it is allowed to be moved. As a result, the action of $R4$ is evaluated to `Y`.

[1] See page 109 for the meaning of symbols &, |, $\sim$ and =>.

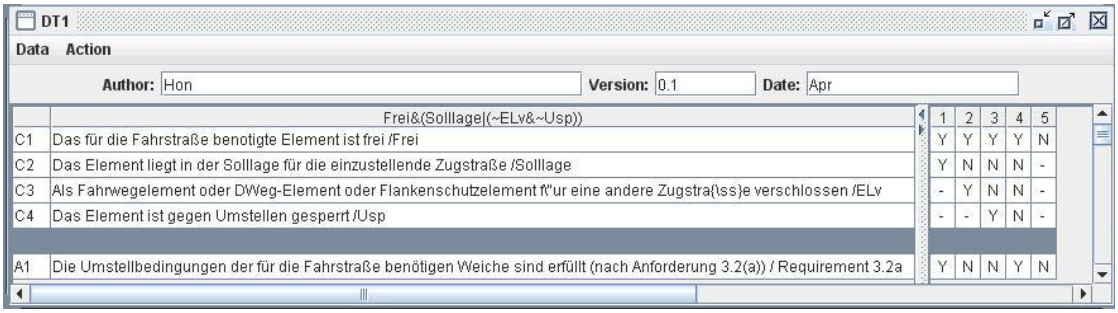


Figure 4.3: The specification of system requirement 3.2(a) in a decision table

As it has been illustrated in Figure 4.3, when a static condition of the interlocking logic is specified in a decision table, the corresponding system requirements are explicitly stated. It cannot lead to any misunderstanding. Furthermore, discussions among different professions of the system development teams become more productive because the system requirements are less difficult to be analyzed. With the help of defined conditions and attributes of objects, the efficiency of the design and the implementation process can be increased. Finally, with functions of checking consistency and completeness w.r.t. the specified conditions of decision tables in the OBLH tool (see Section 5.6.1 and Section 5.6.2), railway engineers can also specify the interlocking logic formally without the help from formal experts. Another assistance for the specification of static conditions are the OBLH methods 2 and 3. The steps of these methods guide users in translating the natural language written checking conditions to consistent static conditions of the state of a safe route. These methods are further elaborated in Section 4.3.4.

A detailed definition of decision tables, the relationship between propositional formulas and decision tables can be found in Chapter 5.

4.2.2.2 Dynamic Conditions

Dynamic conditions are used to define the possible states of a requested safe route and the relationships between these states. As it has been mentioned above, the interlocking logic in [Tut06] defines procedures of a safe route development and possible statuses of a safe route. In OBLH, they are considered as the states of a requested safe route. In the specification [Tut06], a number of states of safe routes has been defined.

ZPZ positive (German: ZPZ positiv), FÜMBli (German: Fahrstraßenfestlegeüberwachungsmelder Blinklicht) and FÜMR (German: Fahrstraßenfestlegeüberwachungsmelder Ruhelicht) are some of the defined statuses of a safe route. Static conditions are defined for these states, for example, when a requested safe route reaches the state ZPZ positive, then route and overlap elements of the requested safe route must be reserved for this requested safe route. In the specification, the logic of the GUI is also defined for this type of states. For example, a green band must be shown in the GUI of the train director when the safe route

reaches the state ZPZ positive (see Table 2.5 on page 13). Via the definition of safe route states, a relation between the logic of developing a safe route and the logic of the GUI can then be defined.

Some procedures of a safe route development have been mentioned in the specification, in which an RIS applies the corresponding defined static conditions to evaluate the infrastructure elements of the requested safe route, such that a requested safe route can proceed to another state. For example, ZPZ is a procedure of a safe route development. One of the static conditions of ZPZ is that each element of the requested safe route is not allowed to be blocked for train movements (see [S13, 2.2.2] in the specification [Tut06]). In this work, if the requested safe route fulfills the static conditions of ZPZ, then the evaluation of ZPZ of this requested safe route is said to be positive. If it does not fulfill these conditions, then ZPZ of this requested safe route is said to be evaluated to be negative.

The relationships between these states define the life cycle of safe routes. In other words, the previous and the next states of each state need to be defined. A definition of these states answers the questions as follows:

- What is the state of the safe route before the checking conditions are applied?
- When an RIS is allowed to use the specific set of checking conditions to evaluate a safe route?
- What is the following state of a safe route if it satisfies a specific set of checking conditions?
- What is the following state of a safe route if it does not satisfy a specific set of checking conditions?

The relationships of states that have been defined in Chapters 2 and 3 of the specification [Tut06] are shown in Table 4.4. UMBP is the abbreviation of the German phrase, Prüfung der Umstellbedingungen. The English meaning of this word is the examination of the checking conditions for moving points.

Based on this analysis of Table 4.4, some ambiguous interpretations of the states of a safe route exist in the specification. For example, the next state of the requested safe route is not defined if a point of the route does not fulfill the corresponding conditions for moving points. It is also uncertain in which states of the safe route a point is set to the requested position. The implementation team can implement an interlocking system where a point of a safe route is set without checking static conditions that have been defined for ZPZ because the corresponding states' relations have not been clearly defined in the specification. However, this implementation might not satisfy the safety requirements of safe route development or the expected behavior of the RIS. If the expected behavior of the RIS is to set a point of the requested safe route after ZPZ of this route is evaluated to be positive, then this implementation error can only be found during the later phases of the system development process. This costs time and resources.

In fact, an interlocking system requirements specification is written in the early phase of a system development, less implementation or design issues should be in-

Chapter	Defined State	Previous State	Next State
2	ZPZ	RIS commences to ZPZ when the command of developing a safe route is input	The static conditions of ZPZ are fulfilled: ZPZ positive not fulfilled: ZPZ negative
	ZPZ positive	ZPZ	-
	ZPZ negative	ZPZ	-
3	UMBP for points located within the route	-	The static conditions are fulfilled: - not fulfilled: -
	UMBP for points points located within the overlap	-	The static conditions are fulfilled: - not fulfilled: -
	UMBP for points provide flank protection	-	The static conditions are fulfilled: - not fulfilled: -

Table 4.4: Defined states of a safe route in LH-ESTW-R

cluded, such that enough freedom is provided to the corresponding system development teams to match their techniques for building an RIS, for example, choosing the sequence for setting the route, overlap and flank protection elements of the safe route in the proper position after the result of ZPZ is evaluated to be positive. However, any unclearness of dynamic conditions which are related to safety requirements shall not happen in system requirements of safety critical systems, like RIS.

The formal specification language for defining dynamic conditions has not been investigated in this work. This formal specification language must satisfy the following requirements:

1. A mechanism to define the states of safe routes must be provided. The states' relationships that are related to safety requirements must be clearly expressed, such that the design or implementation of an RIS satisfies these requirements. This reduces the number of errors that need to be amended in the later phases of the system development process.

2. It is possible to specify implementation or design issues that are not related to safety requirements, in such a way that certain design's freedom is given to system developing companies to suit their technology. And unclear interpretations must be avoided at the same time.

3. A mechanism to find inconsistencies in the dynamic conditions must also be provided. For example, an overlapping overlap is allowed in ZPZ. The static conditions of FÜMR must also state that if the reserved point is without a

movable frog and the requested overlap is a trailing point movement, then the point needs not to be locked. Otherwise, this requested overlap cannot be developed.

4. The modeling language must be easily understood and suitable to be applied by engineers.

One of the possible specification languages for defining dynamic conditions are state machines [EW00, Dou99]. State machines are a design-oriented modeling language [Mil06]. Many implementation issues need to be well-defined in a state machine [Hon06]. However, only one from many different implementation choices can be expressed in a state machine, for example, the sequence of setting the route, overlap and flank protection elements in a proper position for the requested safe route. An RIS can be implemented to set the corresponding elements in a sequence or to set them all at the same time in parallel. And only one of them can be expressed in a state machine. If a state machine is used as the specification tool in this early stage of the system development, system designers will automatically use the defined sequences of the state machine for the design of the system. An interlocking system requirements specification is written with an intention to leave design issues open. Therefore, using a state machine for specifying an interlocking system requirements specification is not suitable. Furthermore, to check the correctness of a state machine, one needs to define safety requirements in temporal logic. However, specification of safety requirements in temporal logic is easily prone to error [Bit02]. It is also uncertain, whether the existing verification methods of state machines can check the mentioned inconsistencies of interlocking system requirements.

Since the specification of dynamic conditions is still under investigation, no corresponding supportive function has been defined and implemented in the OBLH tool.

4.3 Analysis and Verification in Object-Based Lastenheft

The target users of this formal framework are not only formal experts and computer scientists, but also railway engineers. Therefore, a number of ideas is designed to assist different users to apply the concepts of this framework.

These ideas are elaborated in this section, the transformation of formulas and situational analysis in OBLH are first introduced. The function of checking completeness w.r.t. the given conditions of a decision table and consistency of a decision table is then elaborated. Checking the correctness of formal interlocking system requirements and the developed OBLH methods are discussed in the final section.

4.3.1 Transformation of Formulas

The importance of writing precise interlocking system requirements specifications has been discussed in different chapters of this thesis. In Section 4.2.2.1, it has

been explained that applying propositional logic to specify static conditions of the interlocking logic can achieve this goal.

However, from the experience of applying formal methods to railway domains, specifying the interlocking logic in formal logics, like propositional formulas, is not sufficient to increase the simplicity in apprehending the requirements. Therefore, it is important to provide a chance for team members from different backgrounds to interpret the formally specified system requirements. Another goal of this work is to investigate a semantically equivalent and simple form to represent the meaning of propositional formulas. Railway engineers and computer scientists must be able to utilize and read the chosen form of representation without putting much effort to learn it.

Truth table and Decision tables are used to represent the meaning of propositional formulas. The concept of transformations (see Section 5.5.1) and the implementation technique are mathematically well-defined (see Section 5.4), they are implemented in the OBLH tool. In Figure 4.4, the formal specification of requirement 1.7 ($\neg DWEL(v) \vee Solllage \vee (\neg Spitz \wedge \neg BHSS)$), is transformed to a decision table (see Figure 4.4) and a truth table (see Figure 4.5 on page 42).

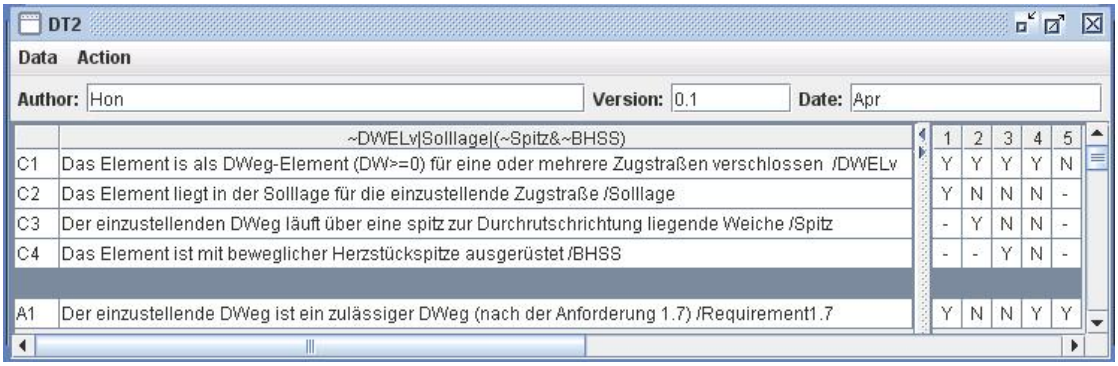


Figure 4.4: The specification of system requirement 1.7 in a decision table

By using the OBLH tool, the propositional formula can be transformed into a decision table, the meaning of the formula can therefore be easily interpreted by different professions in the same way. With this function, complex system requirements in natural language can be well understood in the form of decision tables.

In Section 3.2.4, a number of reasons for the low acceptance of formal methods in the railway domain has been stated. One of them is that railway engineers are not provided with any support to examine formally defined mathematical models. In the OBLH tool, railway engineers can examine a formally defined propositional formula, therefore, their confidence of using formal methods is then increased. Railway engineers can also be involved more in analyzing formal system requirement specifications.

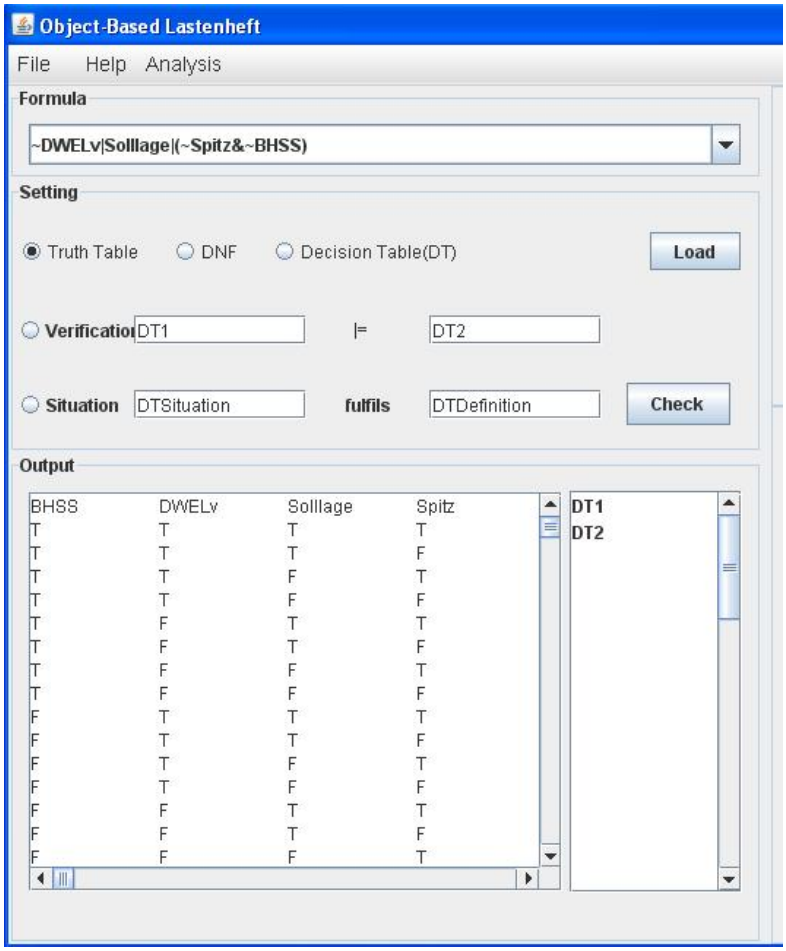


Figure 4.5: The specification of system requirement 1.7 in a truth table

4.3.2 Situational Analysis

Apart from the transformation of propositional formulas to decision tables, there is another way to support analysis of a propositional formula or the corresponding decision table in OBLH. It is the situational analysis.

The idea of this function originates from the working experience with railway engineers. It is common among railway engineers to check whether a given situation satisfies the specified interlocking logic in the interlocking system requirements specification during their work. In other words, the fulfillment of the situation to the system requirements is checked. Normally, the analysis is achieved by first reading the system requirements and then evaluating the situation. For example, given the situation in Figure 2.7 on page 13, railway engineers evaluate whether ZPZ of the requested overlap $DWeg_{Z2}$ is positive.

In OBLH, static conditions of the interlocking logic are expressed in propositional formulas or decision tables. The evaluation of a situation w.r.t. a system requirement can then be achieved by analyzing the defined formula (see Figure 3.1). Another way is to transform the formula into a decision table. Then, one

searches the rule which describes the given situation and the action of this rule is then the evaluation of the situation. This idea is further elaborated in OBLH and the fulfillment of a given situation w.r.t. the system requirements is checked by applying the mathematical concept. This concept will be further elaborated in Section 5.1. The defined operation is also implemented in the OBLH tool (see Section 5.6.3).

The analyses in Figure 3.1 and Figure 3.2 can be obtained by using the OBLH tool. They are illustrated in Figure 4.6 on page 44, Figure 4.7 on page 44, Figure 4.8 on page 45, and Figure 4.9 on page 45, respectively.

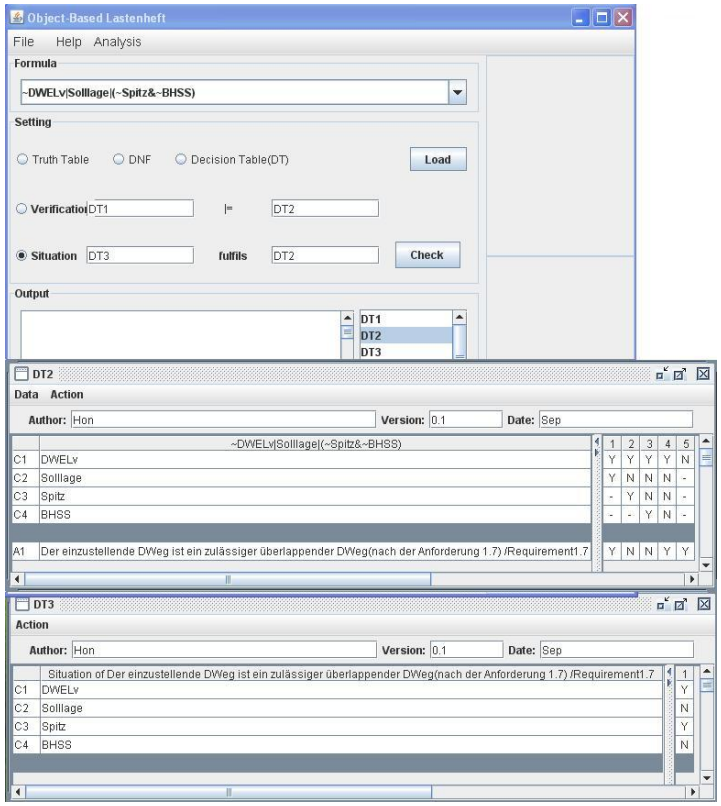


Figure 4.6: Specification of the situation in Figure 3.1

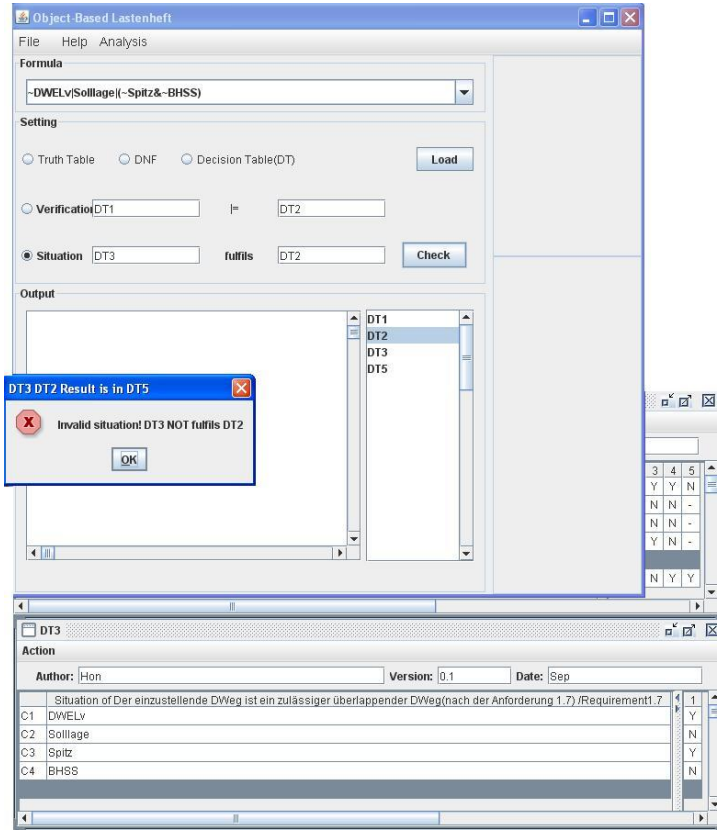


Figure 4.7: Situation analysis of the situation in Figure 3.1

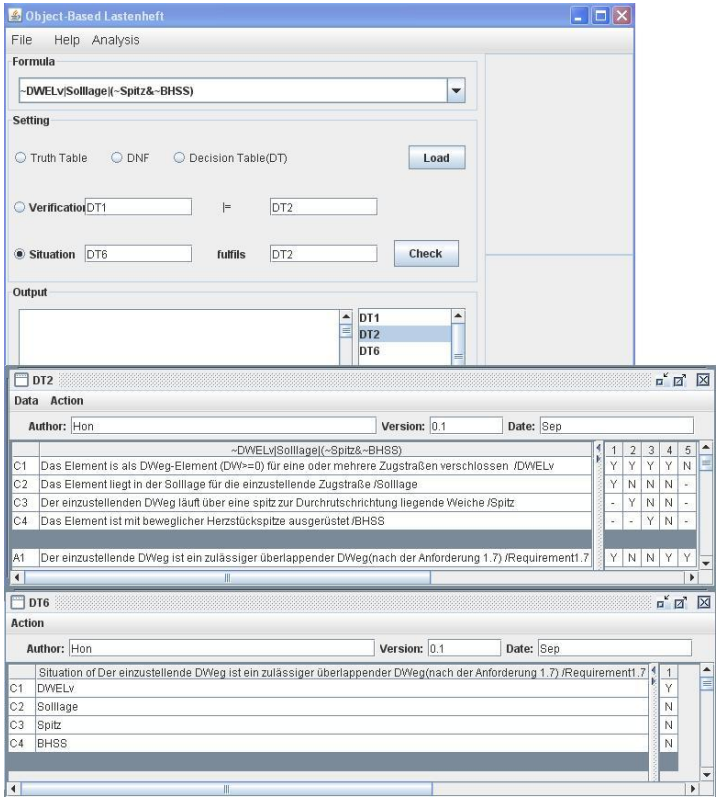


Figure 4.8: Specification of the situation in Figure 3.2

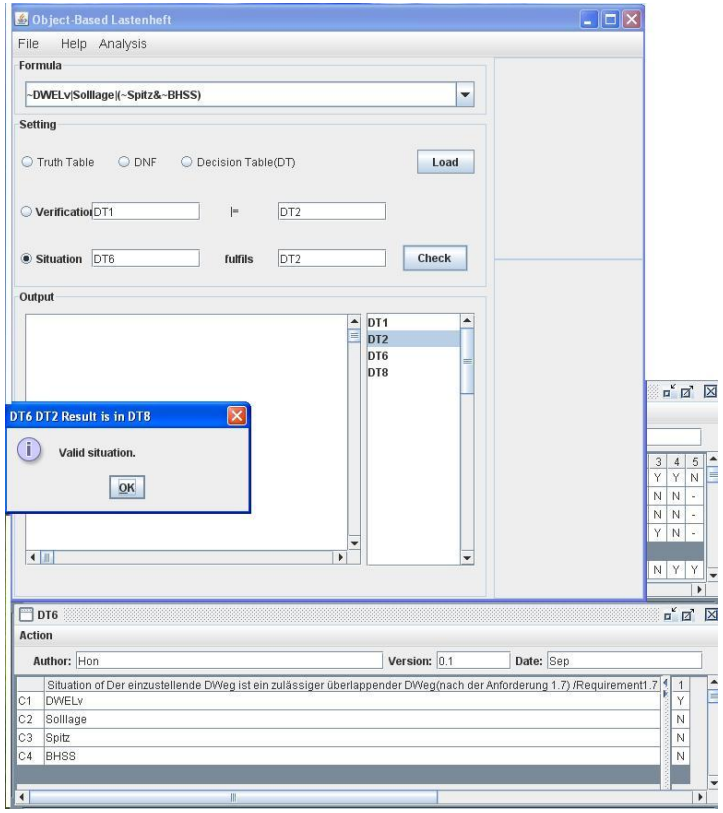


Figure 4.9: Situation analysis of the situation in Figure 3.2

With the help of the situational analysis in the OBLH tool, railway engineers and computer scientists can examine situations and the system requirements or railway concepts quickly.

4.3.3 Completeness and Consistency of Decision Tables

In those projects mentioned in Section 3.2, the formal models and expected behavior of the system are rarely defined by railway engineers. To increase the participation of railway engineers in specification and verification, railway engineers can specify the interlocking logic or expected behavior of the system for checking the correctness of system requirements formally in decision tables in the OBLH tool. These decision tables must satisfy the formal definition of decision tables that is given in Section 5.2.

Based on this definition, a decision table is said to be well-formed in OBLH if it is complete and the specified rules are consistent among each other. Formal definitions of these two properties are given in Section 5.2.2. Informally speaking, if a decision table is used to define a static condition of the interlocking logic and this decision table is incomplete w.r.t. the given conditions, then some situations that need to be evaluated under the system requirement are missing. The decision table in Figure 4.10 defines the requirement 1.7. It is incomplete because a number of possible situations are missing. As a result, some of the situations like the one described in Figure 2.7 on page 13 cannot be evaluated based on this decision table.

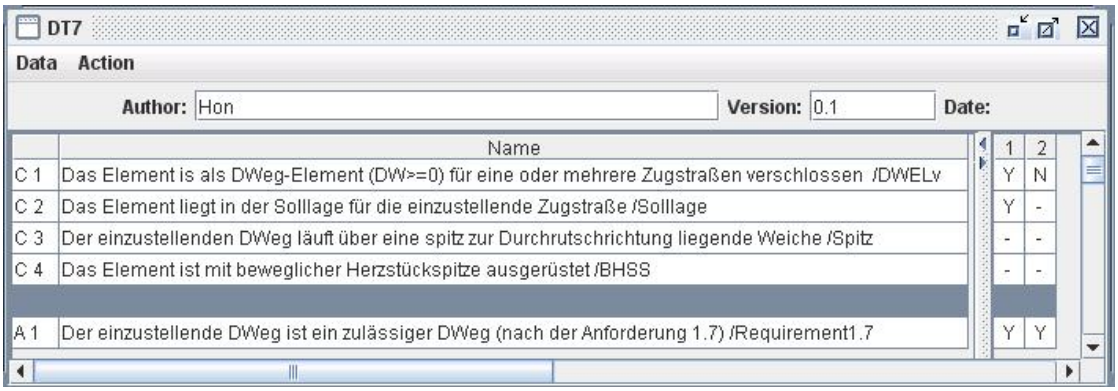


Figure 4.10: An incomplete specification in a decision table

Another important property is the consistency of a decision table. If there are two rules that have the same combination of conditions' values and their evaluations are different in a decision table, then these two rules are said to be inconsistent among each other. For example, the rule $R4$ and $R6$ of the decision table in Figure 4.11 on page 47 are not consistent to each other. They describe the same situation, but their evaluation results are different.

In order to support users to define well-formed decision tables, functions for checking consistency and completeness w.r.t. the defined conditions of a decision

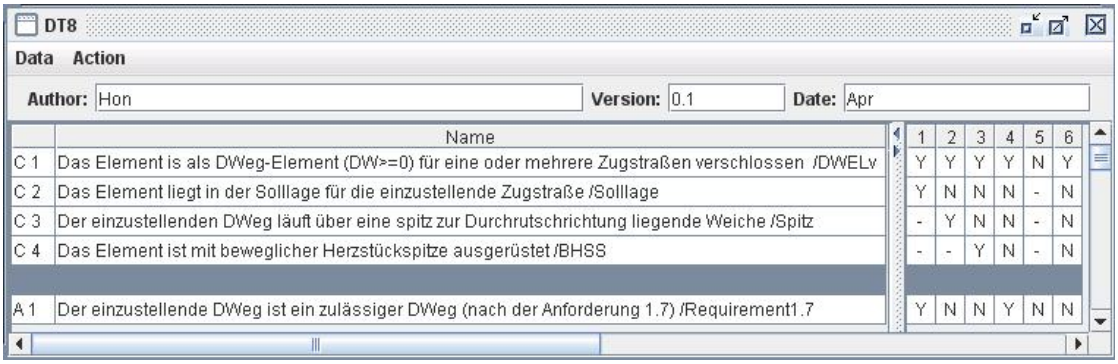


Figure 4.11: An inconsistent specification in a decision table

table are mathematically defined and implemented in the OBLH tool (see Section 5.6.2 and Section 5.6.1). The steps for checking the completeness and consistency of decision tables in Figure 4.10 and Figure 4.11 on page 47 by the OBLH tool are illustrated in Figure 4.12 on page 47 and 4.14, respectively. Figure 4.13 on page 48 and Figure 4.15 on page 49 show the results.

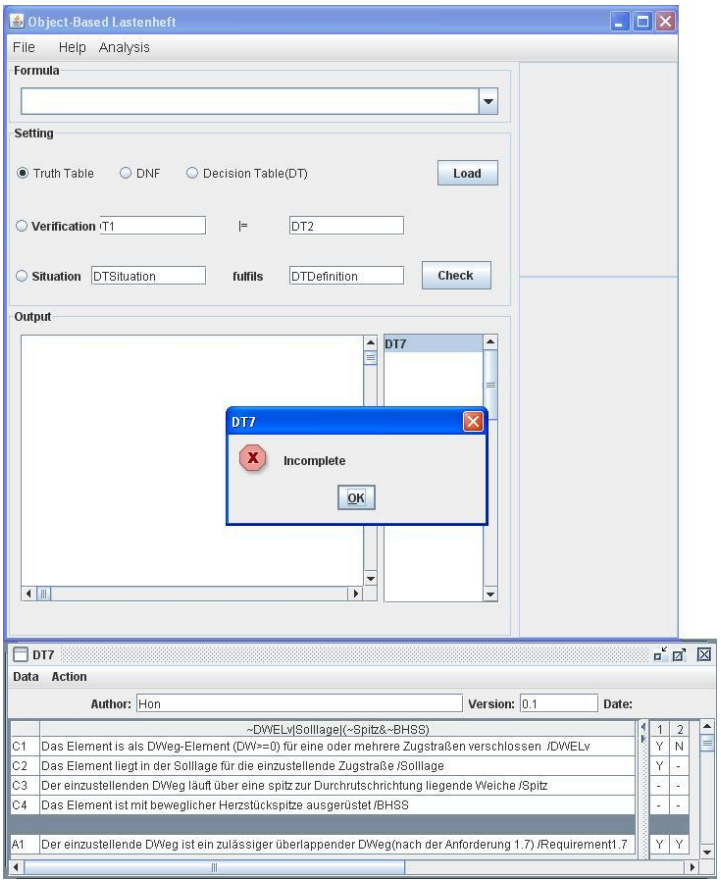


Figure 4.12: Checking the completeness of a decision table in Figure 4.10

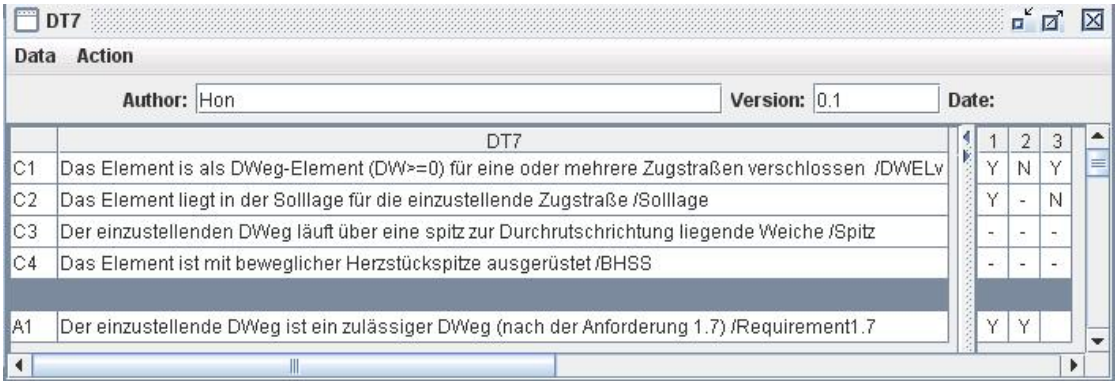


Figure 4.13: The result of the completeness analysis of Figure 4.12

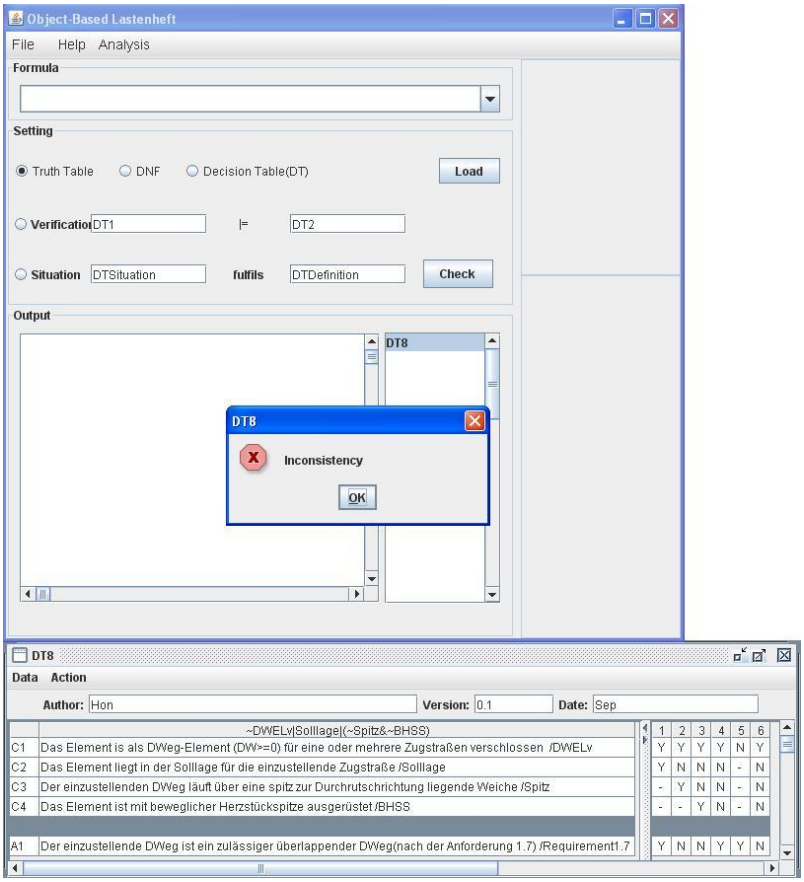


Figure 4.14: Checking the consistency of a decision table in Figure 4.11

With the functions of checking consistency and completeness w.r.t. the specified conditions of decision tables in the OBLH tool, railway engineers are provided with the chance and support to specify the interlocking logic and safety requirements for verification formally and correctly.

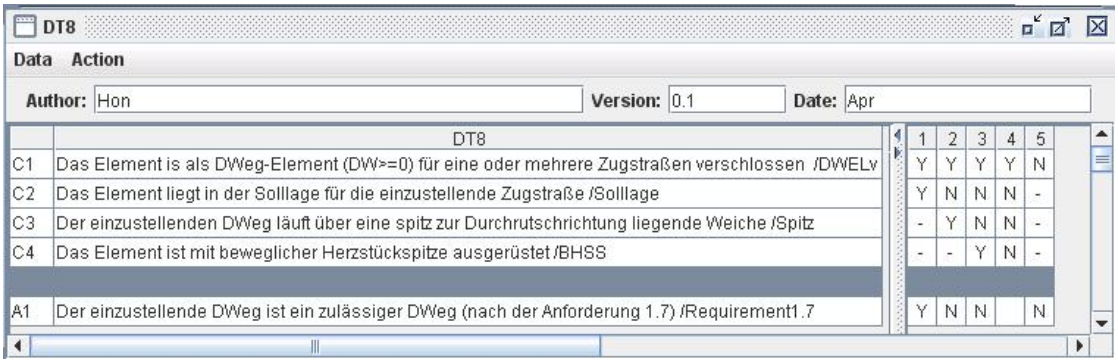


Figure 4.15: The result of the consistency analysis of Figure 4.14

4.3.4 Methods for Specification, Analysis and Verification

Some of the benefits to specify the interlocking system requirements formally based on the concepts of OBLH have been shown in the above sections. Another advantage is that one can apply OBLH methods to specify, analyze and verify the interlocking logic of an interlocking system requirements specification which is written in a natural language, like [Tut06] or analyze and verify a newly defined one in the OBLH tool. Since the current work concentrates on specifying the static conditions of the interlocking logic, the defined methods are designed to support specification, analysis and verification of system requirements of this type.

The functions of each method are briefly listed as follows:

- Method 1: It is a basic method of the OBLH methods. It is used to check the possible unclearness and inconsistency of two system requirements by comparing their evaluation results.

Methods 2, 3 and 4 define usages of method 1 based on their specific context. Method 2 and Method 3 are used to obtain consistent and formal system requirements from a specification that is written in a natural language. Method 4 is used to check the correctness of formal system requirements.

- Method 2: To summarize in natural language written checking conditions specified within one view (e.g. ZPZ conditions for a safe route and in German: ZPZ-Bedingungen für die gesamte Zugfahrstraße) into a consistent well-formed formula or well-formed decision table.

- Method 3: To obtain the consistent static conditions of a state of the safe route (e.g. ZPZ) from in natural language written checking conditions that are defined in two different views (e.g. ZPZ conditions for a safe route, ZPZ conditions for an infrastructure object located within the route and in German: ZPZ-Bedingungen für die gesamte Zugfahrstraße, ZPZ-Bedingungen je Fahrwegelement im Fahrweg).

- Method 4: To verify the correctness of system requirements by comparing them with the expected behavior or safety requirements of the system.

Each of these methods is first elaborated and examples are given to illustrate the steps of each method. The methods are elaborated in detail as follows:

Method 1

Given two system requirements that define similar or the same concepts, inconsistency and unclearness might exist among the requirements. In other words, the evaluation results of these two system requirements over the same situation are different (See Example 9 on page 52). The different evaluations between two system requirements are found by the following steps:

1. Given the system requirements expressed in the forms of well-formed decision tables or well-formed propositional formulas.

2. If system requirements (from step 1) are defined as well-formed formulas, then transform the formulas to decision tables by using the OBLH tool (e.g. $DT1$ and $DT2$).

3. Use the OBLH tool to compare these decision tables by checking whether those situations that are evaluated to `Y` in $DT1$, are also evaluated to `Y` in $DT2$ ($DT1 \models DT2$)[2]. The inconsistent evaluations between the decision tables are indicated in a resulting decision table that is produced by the OBLH tool. In this decision table, those rules with the action value `N` indicate these differences. Users analyze this result.

4. Use the OBLH tool to compare these decision tables by checking whether those situations that are evaluated to `Y` in $DT2$, are also evaluated to `Y` in $DT1$ ($DT2 \models DT1$)[3]. The different evaluations between the decision tables are indicated in a resulting decision table that is produced by the OBLH tool. In this decision table, those rules with the action value `N` indicate these differences. Users analyze this result.

Method 2

Method 2 is applied to summarize the system requirements (checking conditions of a safe route) into a consistent well-formed formula or well-formed decision table within one view or aspect. The system requirements within one view must be fulfilled by the infrastructure elements of the requested safe route. These system requirements consists of the logical and relationship among each other (see Example 10 on page 56).

1. Translate each system requirement (from the set of requirements) which is written in natural language into a well-formed formula or well-formed decision table.

[2]Mathematically, checking whether the admissible situations under a specification $Specification_1$, are also admissible under another specification $Specification_2$, is defined as $Specification_1 \models Specification_2$. See Section 5.1.3 for a detailed elaboration.

[3]The reason of this comparison is given formally in Section 5.1.3.

2. During the translation, if there exists a common attribute in two system requirements and these requirements have the logical and relationship, then it is important to analyze whether inconsistent specifications exist among these two requirements.

3. The analysis of two requirements can be achieved by using Method 1. Choose one of the requirements as a basis and correct the corresponding differences between the system requirements based on the result of the analysis from Method 1.

Method 3

1. Specify the consistent well-formed propositional formulas or well-formed decision tables, respectively, of each view based on Method 2.

2. Compare the consistent well-formed propositional formulas or well-formed decision tables, respectively, of the two different views based on Method 1.

Method 4

The interlocking logic of system requirements defines all the admissible situations in which a safe route can be issued, developed and released. They must also be specified to be admissible in safety requirements or the expected behavior of the system (see Example 12 on page 63). This can be achieved by the following steps:

1. Given the system requirements and checking conditions for verification (e.g. safety requirements) in well-formed decision tables or well-formed propositional formulas, respectively.

2. If system requirements or checking conditions for verification are defined as formulas, then transform the formulas to decision tables by using the OBLH tool (e.g. $DT1$ defines system requirements and $DT2$ specify the checking conditions for verification).

3. Use the OBLH tool to compare these decision tables by checking whether those situations that are evaluated to `Y` in $DT1$, are also evaluated to `Y` in $DT2$ ($DT1 \models DT2$). The differences between the decision tables are indicated in a resulting decision table that is produced by the OBLH tool. In this decision table, those rules with the action value `N` indicate these differences. Users analyze this result.

Examples to illustrate the applications of OBLH methods are provided in the following:

Example 9

Example 4 on page 12 shows that ZPZ conditions for evaluating overlapping overlaps are defined in the system requirement 2.2.1 and system requirement 2.4.1 of [Tut06]. It is then important to analyze whether the evaluation results of the same situation under these two different system requirements are the same. Method 1 can be used to find out the differences. The steps are then illustrated in this example as follows:

1. The system requirements are first translated to propositional formulas:

$$Requirement_{2.2.1} \equiv \quad DWEL(b) \rightarrow (Solllage \vee (\neg BHSS \wedge \neg Spitz))$$

$$Requirement_{2.4.1} \equiv \quad \neg BHSS \rightarrow (Solllage \vee (\neg Usp \wedge \neg(FWEL(b) \vee DWEL(b) \vee FLEL(b))))$$

   $FWEL(b)$: The object is locked or reserved as a route element of other routes.
   Das Element ist als Fahrwegelement für eine andere Fahrstraße beansprucht.

   $DWEL(b)$: The object is locked or reserved as an overlap element of other routes.
   Das Element is als DWeg-Element (DW≥0) für eine oder mehrere Zugstraßen beansprucht.

   $FLEL(b)$: The object is locked or reserved as a flank protection element of other routes.
   Das Element ist als Flankenschutzelement für eine oder mehrere Fahrstraßen beansprucht.

2. The formulas are transformed to decision tables in the OBLH tool. Figure 4.16 shows the decision table of $Requirement_{2.4.1}$ ($DT1$) and the decision table of $Requirement_{2.2.1}$ ($DT2$).

3. The decision table of the system requirement 2.2.1 ($DT2$) is then checked against the decision table of the system requirement 2.4.1 ($DT1$). This step analyzes whether there exists a situation in which ZPZ of the requested overlap $DWeg_{Zx}$ is evaluated to be positive under the definition of $Requirement_{2.2.1}$, while it is considered to be negative under the specification of $Requirement_{2.4.1}$. Figure 4.16 shows the result of $Requirement_{2.2.1} \models Requirement_{2.4.1}$. The resulting decision table $DT4$ contains four rules which indicate the inconsistent evaluations between $Requirement_{2.2.1}$ and $Requirement_{2.4.1}$. The analysis of this result is written as follows:

   a. $R4$ of $DT4$: the corresponding rule in $DT1$ is $R4$ and in $DT2$ it is $R4$. $R4$ of $DT2$ indicates that overlapping overlaps are allowed based on

$Requirement_{2.2.1}$ and they are not allowed in $DT1$. $R4$ of $DT1$ denotes that if an object is not set in a proper position for $DWeg_{Zx}$ ($\neg Solllage$) and it is locked or reserved as an overlap element ($DWEL(b)$), then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. This indicates that no overlapping overlap is allowed under the specification of $Requirement_{2.2.4}$. It is then unclear whether overlapping overlaps are permissible in the specification [Tut06].

b. $R7$ of $DT4$: the corresponding rule in $DT1$ is $R3$ and in $DT2$ it is $R5$. $R3$ of $DT1$ indicates that if an object within $DWeg_{Zx}$ is a manually locking point (Usp) and it is already set in an improper position, then ZPZ of $DWeg_{Zx}$ is evaluated to be negative, while no statement has been specified for this situation in $Requirement_{2.2.1}$. Therefore it is unclear, whether ZPZ of $DWeg_{Zx}$ should be evaluated to be positive or negative under this situation. One should then analyze this situation and decide whether this situation should be admissible or not.

c. $R8$ of $DT4$: the corresponding rule in $DT1$ is $R5$ and in $DT2$ it is $R5$. $R5$ of $DT1$ does not allow a point which is in an improper position for $DWeg_{Zx}$ ($\neg Solllage$) and is locked or reserved as a route element of another safe route ($FWEL(b)$). However, $Requirement_{2.2.1}$ did not specify whether the evaluation ZPZ of $DWeg_{Zx}$ should be positive or negative in this situation. Therefore, one should analyze the situation and decide whether this situation should be allowed or not.

d. $R10$ of $DT4$: the corresponding rule in $DT1$ is $R6$ and in $DT2$ it is $R5$. $R6$ of $DT1$ does not allow a point which is in an improper position for $DWeg_{Zx}$ ($\neg Solllage$) and is locked or reserved as a flank protection element of another safe route ($FLEL(b)$). However, the system requirement 2.2.1 did not specify whether ZPZ of $DWeg_{Zx}$ should be evaluated to be positive or negative in this situation. Therefore, one should analyze this situation and decide the corresponding evaluation of ZPZ of $DWeg_{Zx}$.

4. The decision table of the system requirement 2.4.1 ($DT1$) is then checked against the decision table of the system requirement 2.2.1 ($DT2$). This step analyzes whether there exists a situation in which ZPZ of the requested overlap $DWeg_{Zx}$ is evaluated to be positive under the definition of $Requirement_{2.4.1}$, while it is considered to be negative under the specification of $Requirement_{2.2.1}$. Figure 4.17 shows the result of $Requirement_{2.4.1} \models Requirement_{2.2.1}$. The resulting decision table $DT3$ contains a rule which indicates the inconsistent evaluation between $Requirement_{2.4.1}$ and $Requirement_{2.2.1}$. The analysis of this result is written as follows:

   $R2$ of $DT3$: the corresponding rule in $DT1$ is $R1$ and in $DT2$ it is $R2$. $R2$ of $DT2$ indicates that if the point is equipped with a movable frog it is not set in a proper position for $DWeg_{Zx}$ ($\neg Solllage$) and it is locked for another safe route as an overlap element ($DWEL(b)$). Therefore, it does not

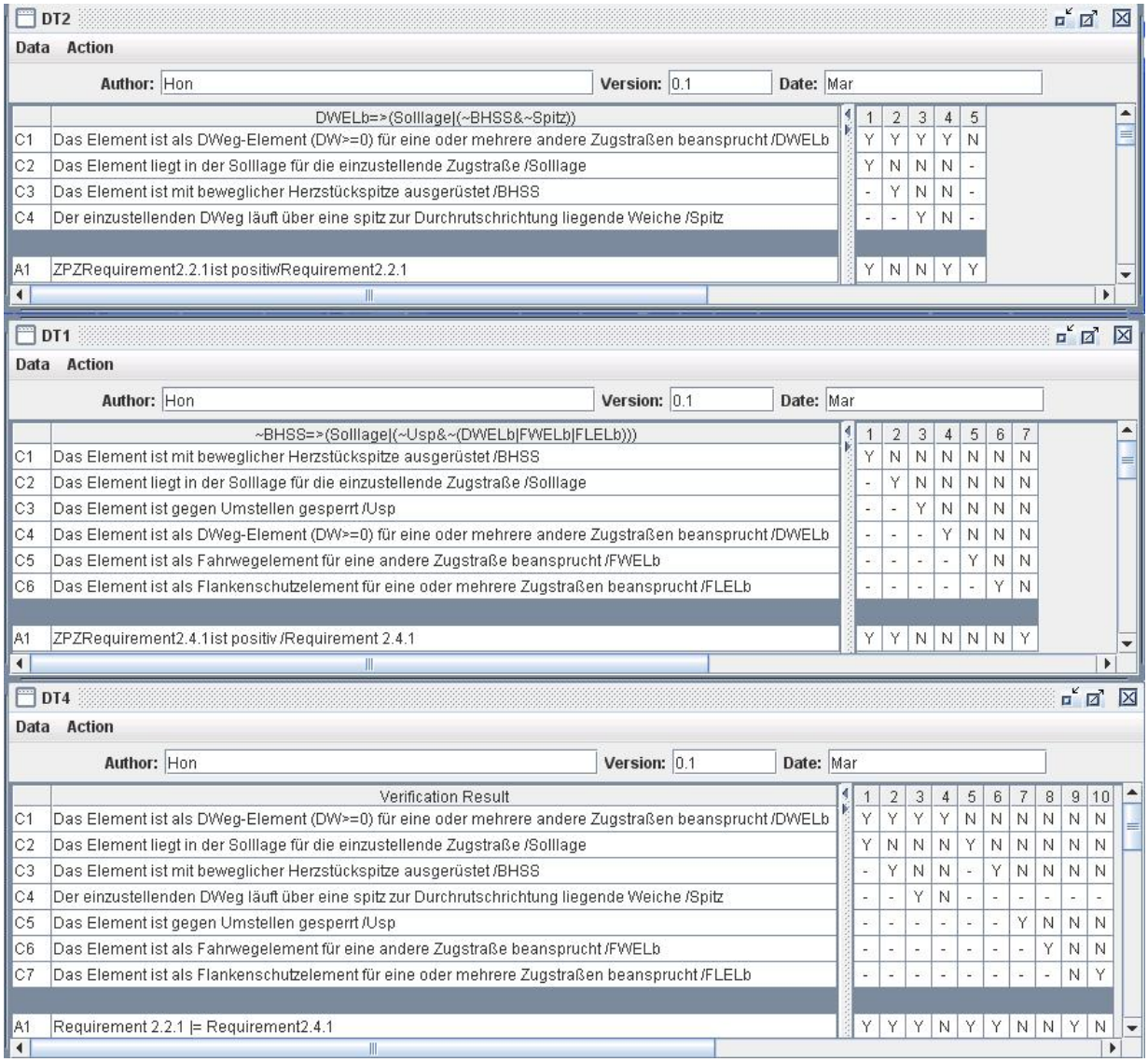


Figure 4.16: Consistency analysis of $Requirement_{2.2.1} \models Requirement_{2.4.1}$

fulfill ZPZ conditions under the specification of $Requirement_{2.2.1}$. However, Requirement 2.4.1 denotes that this situation is allowed. In the natural language written requirement 2.4.1, no requirements have been specified for a point which is equipped with a movable frog in this example. Because of the translation of this requirement into a logical formula it is unclear whether ZPZ of $DWeg_{Zx}$ should be evaluated to be positive or negative. Therefore, one should analyze the situation and decide whether this situation should be allowed or not. ZPZ conditions for a point with a movable frog are specified in the system requirement 2.4.2 of [Tut06]. It indicates that this situation is not allowed.

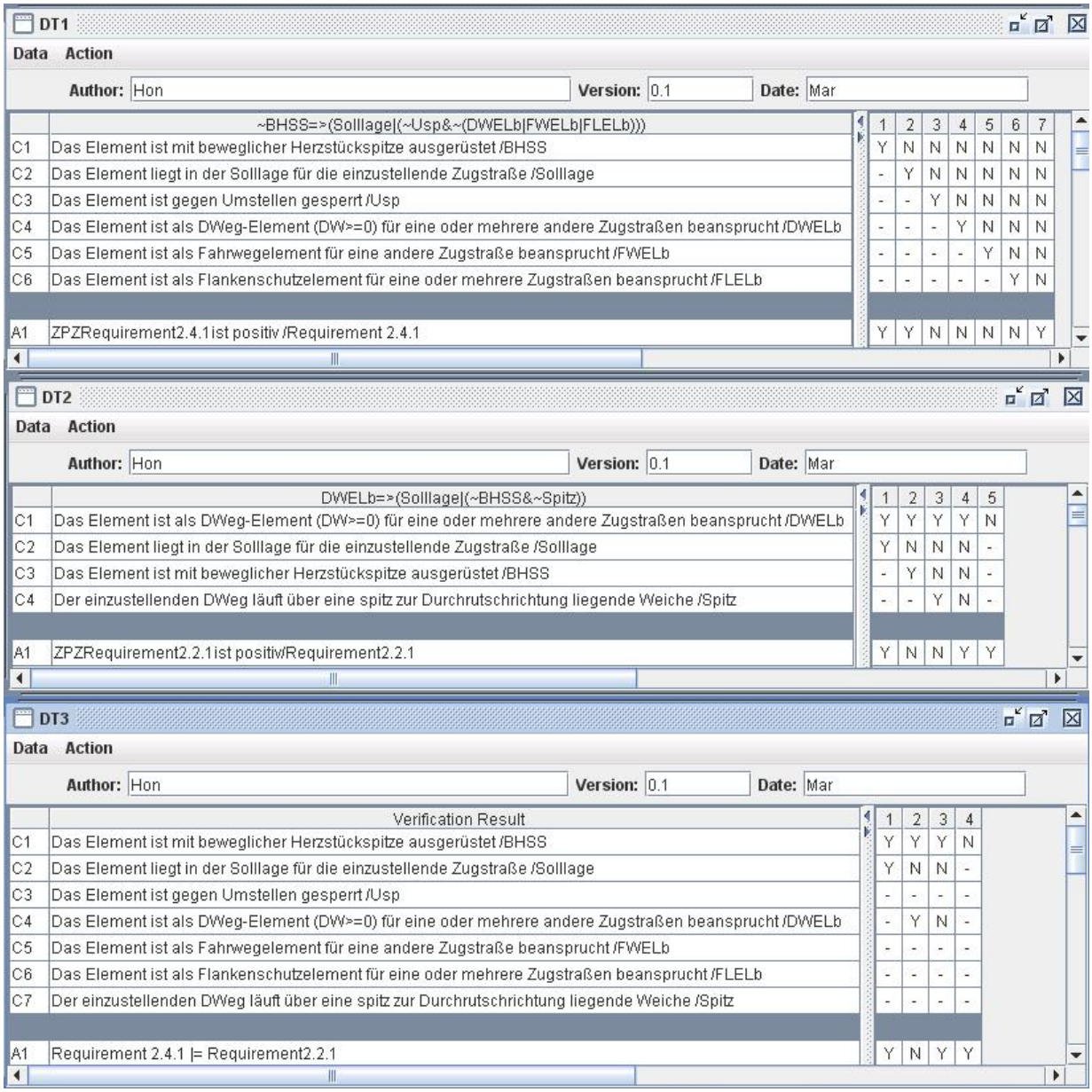


Figure 4.17: Consistency analysis of $Requirement_{2.4.1} \models Requirement_{2.2.1}$

Example 10

To illustrate the concept of Method 2, developing the checking conditions of ZPZ to evaluate a requested overlap within a view (view of a safe route) is shown in this example. For reasons of simplicity, only system requirements 2.2.1 and 2.2.1(a) of the specification are used.

1. The first part of the system requirement 2.2.1 and system requirement 2.2.1(a) are specified as a formula $Requirement_{2.2.1Part1}$ and $Requirement_{2.2.1a}$ in table 4.5.

No.	System requirement	Static condition
2.2	*ZPZ-Bedingungen für die gesamte Zugfahrstraße*	
2.2.1	*Grundsatz für ZPZ* *Im Bereich Start - Ziel - DWeg-Ziel darf keine Auflösestörung vorgelegen haben,* *es darf also kein Fahrwegelement (FW-EL) noch für eine andere Fahrstraße verschlossen sein.* *Es darf keine Zielfestlegung oder ein nicht aufgelöstes DWeg-Ziel vorhanden sein.* *Es gelten aber folgende Ausnahmen:* *a) Durchfahrt* *Bei einer Durchfahrt (gleichzeitige Stellung einer Einfahrzugstrae und einer Ausfahr zugstraße) darf Im Bereich Ziel - D-Weg-Ziel darf eine unmittelbar an das Zielsignal anschließende Zugstraße vorhanden sein.*	$\neg FWEL(b) \wedge \neg DWEL(b) \wedge (FLEL(b) \rightarrow Solllage)$ $FWEL(b) \rightarrow ($ $Solllage \wedge EL(Durchfahrt))$

Table 4.5: The specification of system requirements 2.2.1 and 2.2.1(a)

$EL(Durchfahrt)$: The locking or reservation belongs to a part of the requested non-stop safe route
Nutzung gehört zu einem Teil der einzustellenden Durchfahrt

2. The formula $Requirement_{2.2.1a}$ ($Requirement_{2.2.1a} \equiv FEWL(b) \rightarrow (Solllage \wedge EL(Durchfahrt))$) consists of an attribute $FWEL(b)$ that has been used in the formula $Requirement_{2.2.1Part1}$ ($Requirement_{2.2.1Part1} \equiv \neg FWEL(b) \wedge \neg DWEL(b) \wedge (FLEL(b) \rightarrow Solllage)$). These formulas contain the logical and relationship. Therefore it is important to analyze these two formulas by using Method 1.

3. Both formulas are transformed to decision tables in the OBLH tool. $DT5$ and $DT6$ in Figure 4.18 are the decision tables of $Requirement_{2.2.1Part1}$ and $Requirement_{2.2.1a}$, respectively.

4. The decision table of $Requirement_{2.2.1Part1}$ ($DT5$) is first checked against the decision table of $Requirement_{2.2.1a}$ ($DT6$). This step analyzes whether there exists a situation in which ZPZ of the requested overlap $DWeg_{Zx}$ is evaluated to be positive in $DT5$, while it is considered to be negative in $DT6$. Figure 4.18 shows the result of $Requirement_{2.2.1Part1} \models Requirement_{2.2.1a}$. The resulting decision table $DT7$ contains one rules. The action of this rule is evaluated to `Y`. This means, no situation is evaluated to be admissible under $Requirement_{2.2.1Part1}$, while it is evaluated to be inadmissible under the specification of $Requirement_{2.2.1(a)}$.

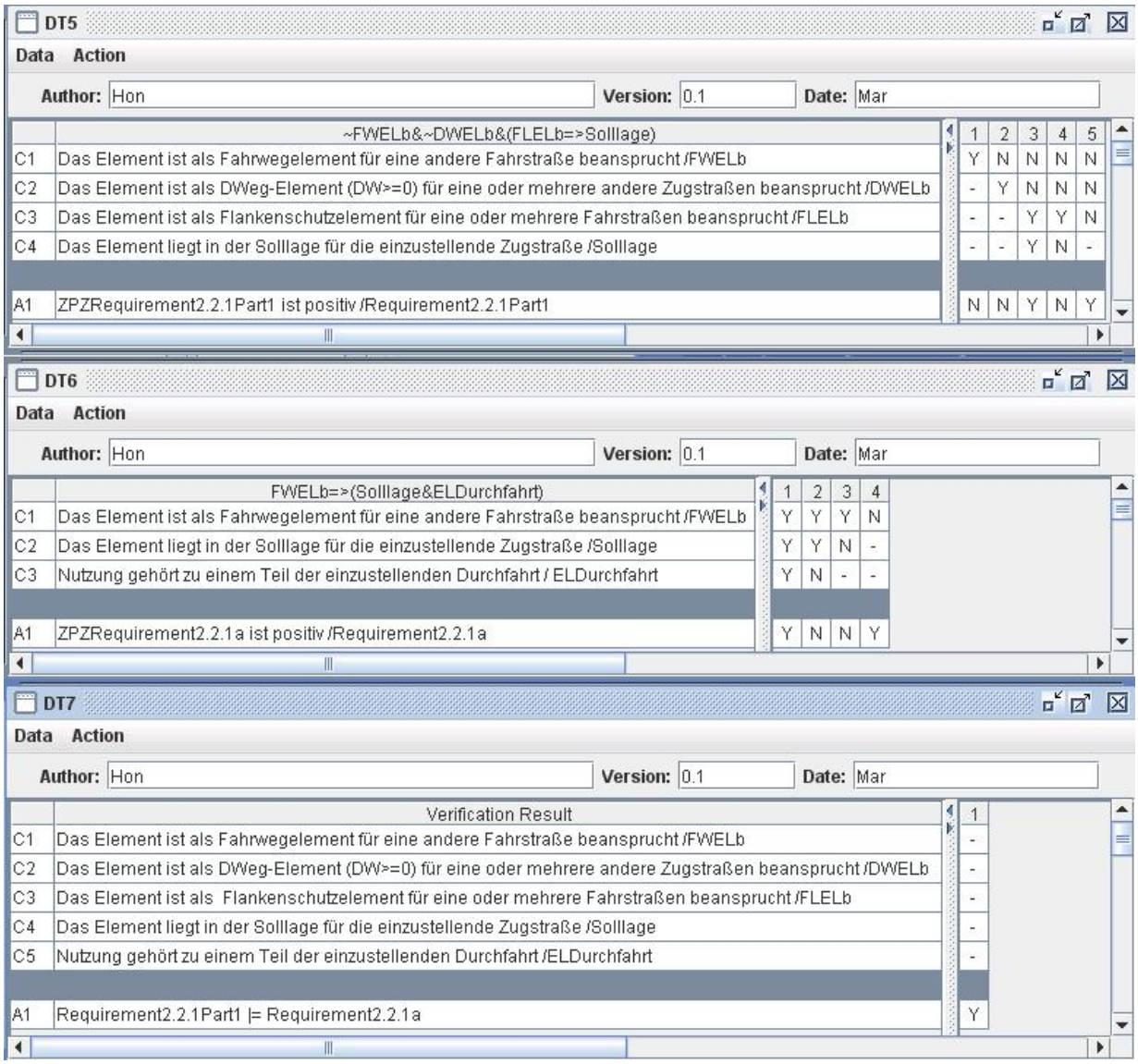


Figure 4.18: Consistency analysis of $Requirement_{2.2.1Part1} \models Requirement_{2.2.1a}$

5. The decision table of $Requirement_{2.2.1a}$ ($DT6$) is checked against the decision table of $Requirement_{2.2.1Part1}$ ($DT5$). This step analyzes whether there exists a situation in which ZPZ of $DWeg_{Zx}$ is evaluated to be positive

in $DT6$, while it is considered to be negative in $DT5$. Figure 4.19 shows the result of $Requirement_{2.2.1a} \models Requirement_{2.2.1Part1}$. The resulting decision table $DT8$ contains four rules which indicate inconsistent evaluations between $Requirement_{2.2.1}$ and $Requirement_{2.2.1Part1}$. The analysis of this result is written as follows:

a. $R1$ of $DT8$: the corresponding rule in $DT5$ is $R1$ and in $DT6$ it is $R1$. $R1$ of $DT5$ indicates that if the object within $DWeg_{Zx}$ is already locked or reserved as a route element of another safe route ($FWEL(b)$), then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. However, the requested safe route is a part of the non-stop route (the entrance route), this object can be locked or reserved for another part of the non-stop route (the exit route). This situation is evaluated to be positive in $R1$ of $DT6$. In other words, a non-stop route cannot be developed by using $DT5$.

b. $R4$ and $R6$ of $DT8$: the corresponding rule in $DT5$ is $R2$ and in $DT6$ it is $R4$. $R2$ of $DT5$ indicates that if the object within $DWeg_{Zx}$ is already locked or reserved as an overlap element of another safe route ($DWEL(b)$), then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. Since no declaration of this situation has been made in $Requirement_{2.2.1a}$ it is unclear whether ZPZ of $DWeg_{Zx}$ should be evaluated to be positive or negative in this situation, one should analyze this situation and decide on the corresponding evaluation of ZPZ of $DWeg_{Zx}$.

c. $R8$ of $DT8$: the corresponding rule in $DT5$ is $R4$ and in $DT6$ it is $R4$. $R4$ of $DT5$ indicates that if the object within $DWeg_{Zx}$ is already locked or reserved as a flank protection element of another safe route ($FLEL(b)$) and it is set in an improper position for $DWeg_{Zx}$ ($\neg Solllage$), then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. The corresponding evaluation of ZPZ of $DWeg_{Zx}$ must be decided based on the result of an analysis.

6. Based on the above differences, the decision table of $Requirement_{2.2.1Part1}$ is chosen as the basis of the correction. The result is shown in Figure 4.20.

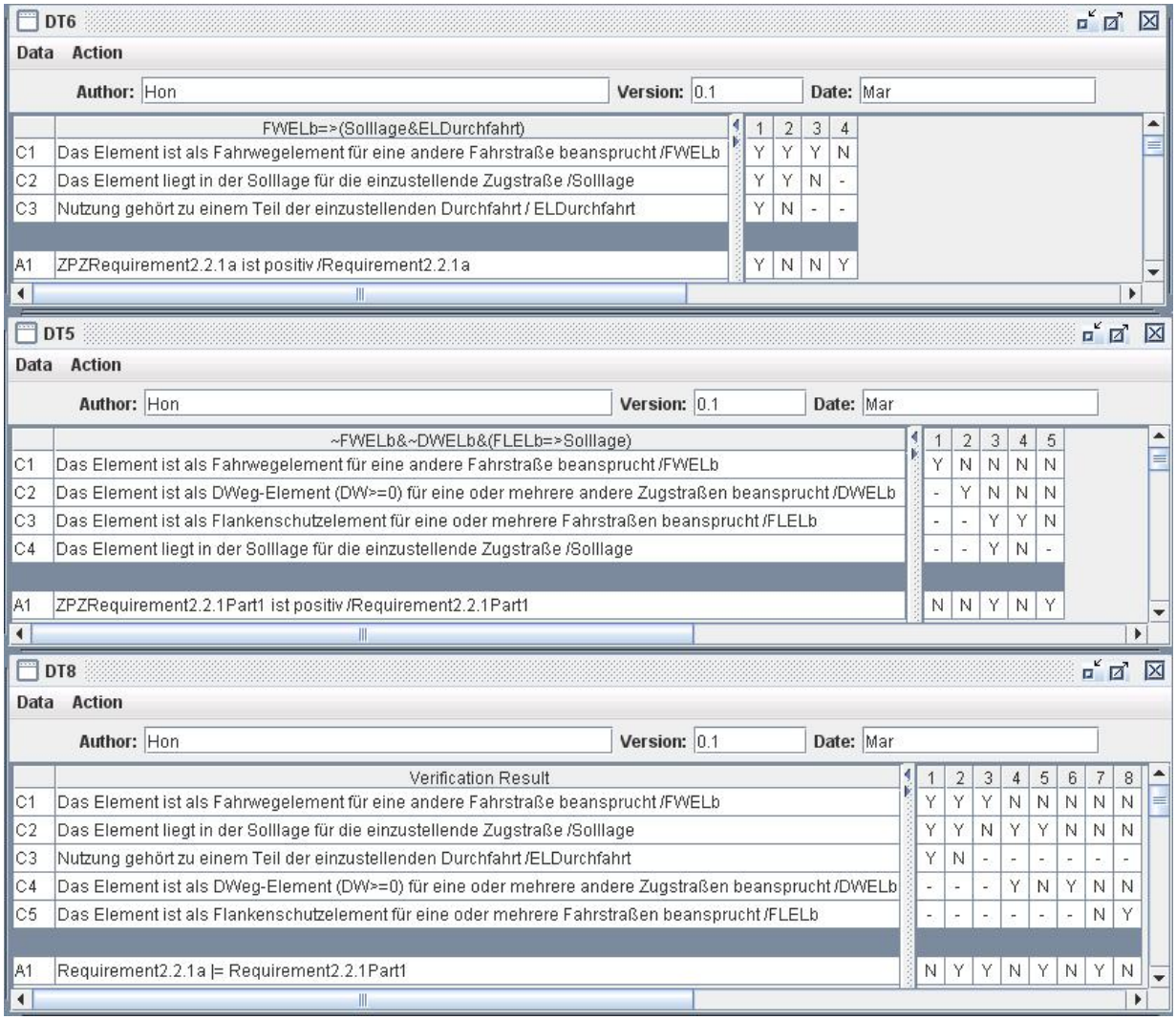


Figure 4.19: Consistency analysis of $Requirement_{2.2.1a} \models Requirement_{2.2.1Part1}$

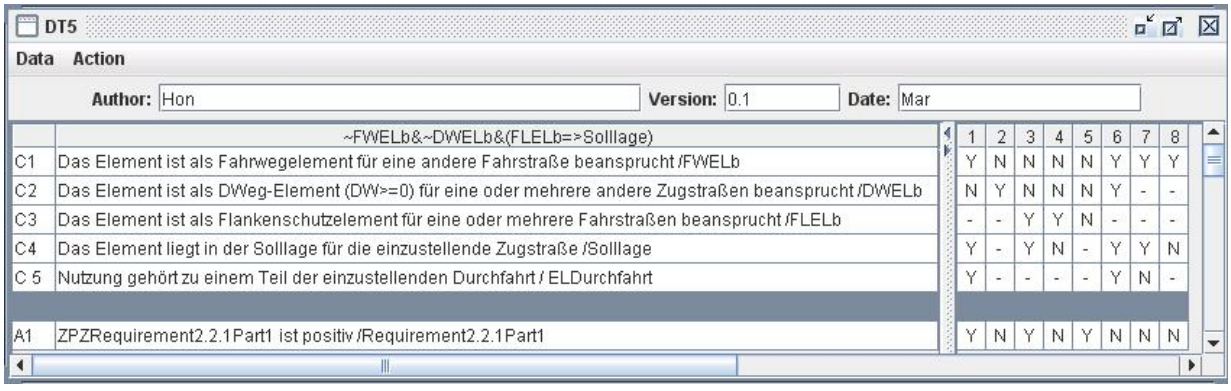


Figure 4.20: The correct decision table

Example 11

Method 3 is used to specify the consistent static conditions of the interlocking logic. In this example, parts of the steps in developing the static condition of ZPZ for evaluating the requested overlap $DWeg_{Zx}$ are illustrated as follows:

1. The formulas of each view are developed based on Method 2 and they are specified as follows:

   $ZPZ_{DWegGesamtzugstraße} \equiv$ $(FWEL(b) \rightarrow (Solllage \wedge EL(Durchfahrt)))$ $\wedge(DWEL(b) \rightarrow (Solllage \vee (\neg BHSS \wedge$ $\neg Spitz))) \wedge (FL(b) \rightarrow Solllage) \wedge \neg Bsp$ $(EL(Durchfahrt) \rightarrow (FWEL(b) \wedge$ $\neg DWEL(b)))$

   $ZPZ_{DWegElement} \equiv$ $Solllage \vee (\neg Usp \wedge \neg(FWEL(b) \vee DWEL(b)$ $\vee FLEL(b))) \wedge (EL(Durchfahrt) \rightarrow$ $(FWEL(b) \wedge \neg DWEL(b)))$

   Bsp: The object is blocked for train movements. Eine Befahrbarkeitssperre ist gesetzt.

   $ZPZ_{DwegElement}$: The evaluation ZPZ conditions of the requested overlap is positive (based on the ZPZ conditions in the view of an object). ZPZ des einzustellenden DWeg ist positiv (nach den ZPZ-Bedingungen auf der Ebene "Fahrwegelement im Durchrutschweg").

   $ZPZ_{DwegGesamtzugstraße}$: The evaluation ZPZ conditions of the requested overlap is positive (based on the ZPZ conditions in the view of a safe route). ZPZ des einzustellenden DWeg ist positiv (nach den ZPZ-Bedingungen für die Ebene "gesamte Zugfahrstraße").

2. The formulas are transformed to decision tables in the OBLH tool as it is shown in Figure 4.21. $DT8$ and $DT9$ are the decision tables of $ZPZ_{DWegGesamtzugstraße}$[4] and $ZPZ_{DWegElement}$, respectively.

3. The decision table of $ZPZ_{DWegGesamtzugstraße}$ ($DT8$) is checked against the decision table of $ZPZ_{DWegElement}$ ($DT9$). This step analyzes whether there exists a situation in which ZPZ of $DWeg_{Zx}$ is evaluated to be positive in $DT8$, while it is considered to be negative in $DT9$. Figure 4.21 shows the result of $ZPZ_{DWegGesamtzugstraße} \models ZPZ_{DWegElement}$. The resulting decision table $DT10$ contains two rules which indicate inconsistent evaluations

[4] $(EL(Durchfahrt) \rightarrow (FWEL(b) \wedge \neg DWEL(b)))$ is not a defined system requirement of the specification [Tut06], it is defined in this work for the purpose of mathematical specifications, such that physically this impossible situation is defined. The evaluation result of this situation must not be admissible. In other words, the corresponding rule of the decision table must have the action value `N` (see Section 6).

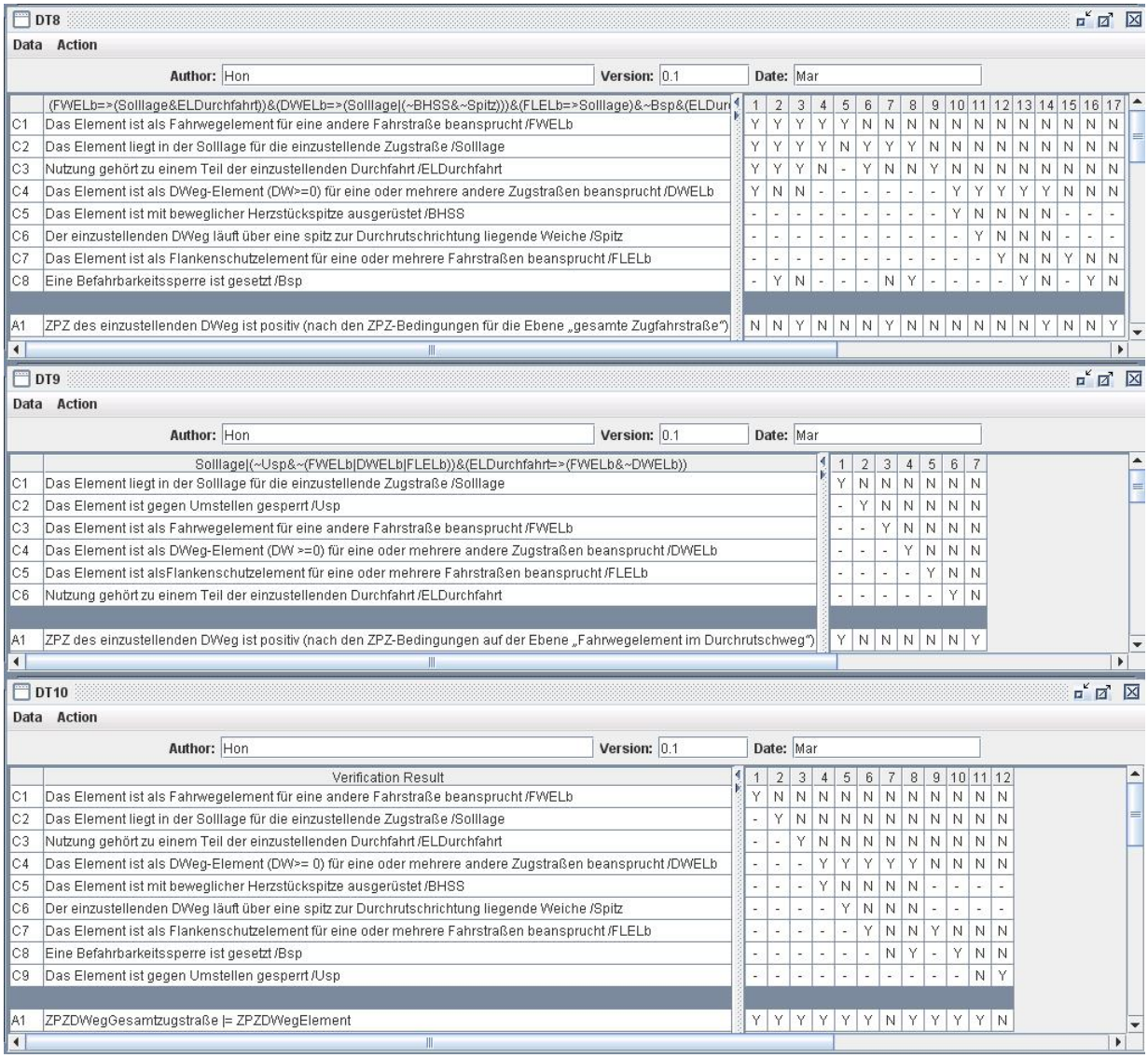


Figure 4.21: Consistency analysis of $ZPZ_{DWegGesamtzugstraße}$ and $ZPZ_{DWegElement}$

between $ZPZ_{DWegGesamtzugstraße}$ and $ZPZ_{DWegElement}$. The analysis of this result is written as follows:

a. $R7$ of $DT10$: the corresponding rule in $DT8$ is $R17$ and in $DT9$ it is $R2$. $R2$ of $DT9$ denotes that if an object is not in a proper position for $DWeg_{Zx}$ ($\neg Solllage$) and it is locked or reserved as an overlap element ($DWEL(b)$), then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. However, $R17$ of $DT8$ indicates that if this object is equipped without a movable frog and the train movement of $DWeg_{Zx}$ is a trailing point movement, then ZPZ of $DWeg_{Zx}$ is evaluated to be positive. In other words, this type of overlapping overlaps cannot be developed under the specification of $ZPZ_{DWegElement}$. It is then ambiguous whether overlapping overlaps are permissible in the specification [Tut06].

b. $R12$ of $DT10$: the corresponding rule in $DT8$ is $R2$ and in $DT9$ it is $R15$. $R15$ of $DT9$ indicates that if the object within $DWeg_{Zx}$ is a manually locking point (Usp) and it is set in an improper position for $DWeg_{Zx}$, then ZPZ of $DWeg_{Zx}$ is evaluated to be negative. Since no declaration of this situation has been made in $ZPZ_{DWegGesamtzugstraße}$ it is unclear whether ZPZ of $DWeg_{Zx}$ should be evaluated to be positive or negative in this situation under this specification, one should analyze the situation and decide the corresponding evaluation of ZPZ of $DWeg_{Zx}$.

4. Based on the analysis of the result of the comparison, $DT8$ is chosen as the basis of correction and the corrected decision table is shown in Figure 4.22.

5. The next step is to compare the decision table of $ZPZ_{DWegElement}$ ($DT9$) with the decision table of $ZPZ_{DWegGesamtzugstraße}$ ($DT8$). The principle of applying this method has been demonstrated in this example, the result of $ZPZ_{DWegElement} \models ZPZ_{DWegGesamtzugstraße}$ is not further discussed in this thesis[5].

	DT5	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
C1	Das Element ist als Fahrwegelement für eine andere Fahrstraße beansprucht /FWELb	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
C2	Das Element liegt in der Solllage für die einzustellende Zugstraße /Solllage	Y	Y	Y	Y	N	Y	Y	Y	N	N	N	N	N	N	N	N	N	N	N	N	N
C3	Nutzung gehört zu einem Teil der einzustellenden Durchfahrt /ELDurchfahrt	Y	Y	Y	N	-	Y	N	N	Y	N	N	N	N	N	N	N	N	N	N	N	N
C4	Das Element ist als DWeg-Element (DW>=0) für eine oder mehrere andere Zugstraßen beansprucht /DWELb	Y	N	N	-	-	-	-	-	-	Y	Y	Y	Y	Y	N	N	N	Y	N	N	N
C5	Das Element ist mit beweglicher Herzstückspitze ausgerüstet /BHSS	-	-	-	-	-	-	-	-	-	Y	N	N	N	N	-	-	-	N	N	Y	N
C6	Der einzustellenden DWeg läuft über eine spitz zur Durchrutschrichtung liegende Weiche /Spitz	-	-	-	-	-	-	-	-	-	-	Y	N	N	N	-	-	-	N	N	-	Y
C7	Das Element ist als Flankenschutzelement für eine oder mehrere Fahrstraßen beansprucht /FLELb	-	-	-	-	-	-	-	-	-	-	-	Y	N	N	Y	N	N	N	N	N	N
C8	Eine Befahrbarkeitssperre ist gesetzt /Bsp	-	Y	N	-	-	-	N	Y	-	-	-	-	Y	N	-	Y	N	N	N	N	N
C9	Das Element ist gegen Umstellen gesperrt /Usp	-	-	-	-	-	-	-	-	-	-	-	-	-	N	-	-	N	Y	Y	Y	Y
A1	ZPZ des einzustellenden DWeg ist positiv (nach den ZPZ-Bedingungen für die Ebene „gesamte Zugfahrstraße“)	N	N	Y	N	N	N	Y	N	N	N	N	N	N	Y	N	N	Y	N	N	N	N

Figure 4.22: The correct decision table

[5]The result of this step has been produced and discussed in a project for the German Rail (German: Deutsche Bahn) [HGS09].

Example 12

As it has been mentioned above, in Method 4, the system requirements are checked against the expected behavior of system requirements or safety requirements. It is only important to check whether those situations which are evaluated to be admissible under the system requirements, are also admitted under the expected behavior of the system. Therefore, Method 4 can be applied. The steps of applying Method 4 for the verification are then illustrated in this example as follows:

1. The system requirement 2.2.1 is specified as a formula $Requirement_{2.2.1}$ (see Example 9 on page 52 for the definition of the formula) and the expected behavior $Requirement_{2.2.1ExpectedBehavior}$ is defined in a decision table $DT3$ (see Figure 4.23).

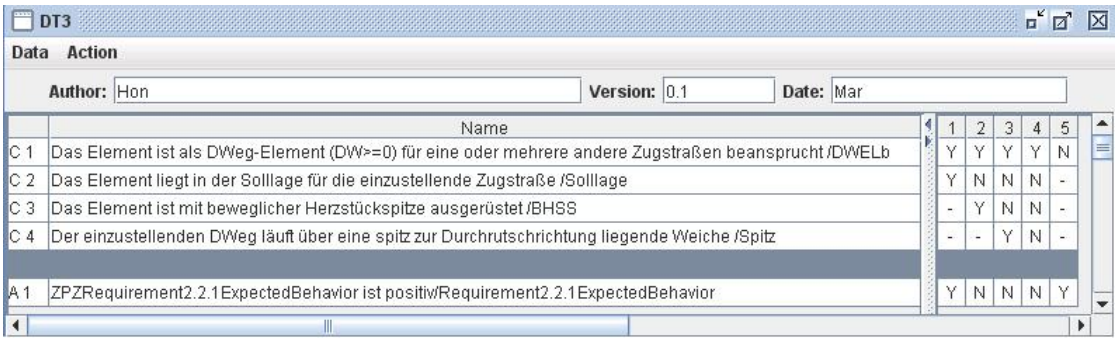


Figure 4.23: The expected behavior of the system requirement 2.2.1

2. $Requirement_{2.2.1}$ is transformed to a decision table $DT2$ with the OBLH tool.

3. The decision table of the system requirement 2.2.1 ($DT2$) is verified against the decision table of the expected behavior ($DT3$). This step analyzes whether there exists a situation in which the evaluation of ZPZ for the requested overlap $DWeg_{Zx}$ is evaluated to be positive under the specification of $Requirement_{2.2.1}$, while it is considered to be negative under the expected behavior. Figure 4.24 shows the result of $Requirement_{2.2.1} \models Requirement_{2.2.1ExpectedBehavior}$. The resulting decision table $DT4$ contains a rule which indicates the inconsistent evaluation between $Requirement_{2.2.1}$ and the expected behavior of this requirement. The analysis of this result is written as follows:

   $R4$ of $DT4$: the corresponding rule in $DT2$ is $R4$ and in $DT3$ it is $R4$. $R4$ of $DT3$ indicates that if the point within $DWeg_{Zx}$ is not in a proper position ($\neg Solllage$) and it is locked as an overlap element for another safe route, then ZPZ of $DWeg_{Zx}$ is evaluated to be negative independent of the construction of this point (e.g. with or without a movable frog). However,

under $Requirement_{2.2.1}$, this situation can be evaluated to be positive if the point is not equipped with a movable frog ($\neg BHSS$). Therefore, there is a situation in which the evaluation result is different under $Requirement_{2.2.1}$ and the expected behavior.

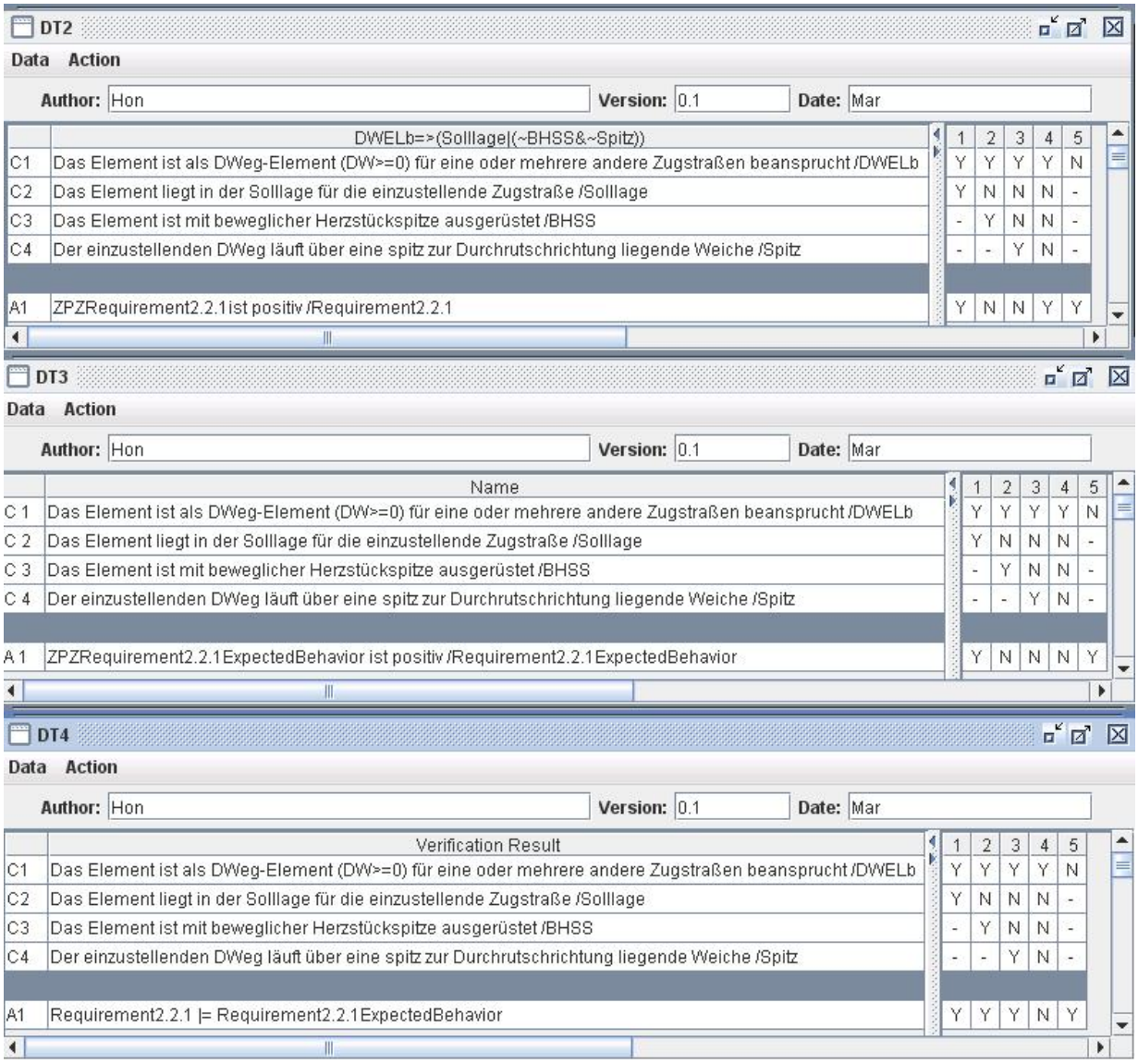


Figure 4.24: Verification of the system requirement 2.2.1 against expected behavior

When the system requirements that are written in a natural language or newly defined system requirements specifications are expressed in the forms of decision tables or propositional formulas, the correctness of interlocking system requirements specifications that can be checked with the OBLH methods. Inconsistencies of specifications that have been mentioned in Section 2.3 can be discovered. Furthermore, the formal static conditions can be verified against the expected behavior of the system and safety requirements, such that the costs of amending

the corresponding discovered incorrectness can be reduced. As it has been shown, railway engineers can also participate in checking the correctness of these system requirements with the usage of decision tables.

Chapter 5

Formal Logics in Object-Based Lastenheft

Concepts of OBLH to support specification, analysis and verification of the interlocking system requirements have been discussed informally in the previous chapter. In this chapter, these concepts are modeled and defined mathematically by propositional logic, such that they can be well understood and correctly implemented. The suitable technique is found to implement these mathematical operations as a program. In this work, the technique Ordered Binary Decision Diagrams (OBDDs) is used.

In this chapter, an introduction to propositional logic is given in Section 5.1. Decision tables are an important form of representing the meaning of interlocking system requirements in OBLH. Therefore, a formal definition of decision tables in OBLH is given in Section 5.2. The relationships between propositional logic, decision tables and the OBDDs technique are elaborated in Section 5.3. The OBDDs technique is formally introduced in Section 5.4. The algorithms to implement the OBLH concepts with this technique are then described in Section 5.5 and Section 5.6.

5.1 Introduction to Propositional Logic

It has been mentioned in Chapter 4 that propositional logic is one of the classical logics to model declarative statements and to assist reasoning [HR04]. In mathematics, each formal specification language comprises two components: syntax and semantics. The syntax includes the definition of notations that can be used in the logic and deduction rules to manipulate the notations. The semantics defines the meaning of notations. With the well-defined semantics and syntax of the logic, making conclusions from stated facts can be achieved. Syntactically, reasoning is achieved by applying the deduction rules of the logic among the conclusions and facts. Semantically, they are carried out by comparing the meaning of the facts and the conclusions.

The syntax and semantics of propositional logic are briefly introduced in Section 5.1.1 and Section 5.1.2. Reasoning in propositional logic is then discussed in

Section 5.1.3.

5.1.1 Syntax

The syntax of propositional logic consists of three components: the symbols that are used in the language, the syntactic rules for combining these symbols and the proof rules for reasoning. These three components can be understood in the sense of a natural language. The symbols are the possible words that can be used. The syntactic rules describe the grammar of the language. Finally, the proof rules are the human logic that is used for making decisions without giving explicit meaning to the facts.

In propositional logic, the fundamental element is a proposition, called a formula. It can be considered as a declarative statement in natural language. In Example 8 on page 19, $Requirement_{1.7}$ has been specified as a complex formula in propositional logic, it is called a complex formula because it is composed of atomic formulas and logical connectives. The atomic formulas are $DWEL(v)$, $Solllage$, $Spitz$ and $BHSS$ and the logical connectives are $\neg$, $\wedge$, $\vee$. The atomic formulas cannot be further decomposed and do not contain any connective. Complex formulas are simply called formulas in this report.

The syntactic part of propositional logic defines the possible connectives that are allowed to be used in propositional logic and also the syntactic rules for forming formulas. These syntactic rules are expressed in **B**ackus **N**aur **F**orm (BNF) [Nau60, ML86][1] as follows:

$\phi ::= q|\neg\phi|\phi \wedge \phi|\phi \vee \phi|\phi \rightarrow \phi|(\phi)$

q is an atomic formula and ϕ is a formula. A propositional formula that is composed based on these six expression rules ($Rule_1$ is q, $Rule_2$ is $\neg\phi$, $Rule_3$ is $\phi \wedge \phi$, $Rule_4$ is $\phi \vee \phi$, $Rule_5$ is $\phi \rightarrow \phi$ and $Rule_6$ is (ϕ)) is called **w**ell **f**ormed **f**ormula (wff). The steps in composing a wff $\neg DWEL(v) \vee Solllage \vee \neg Spitz$ based on these rules are as follows:

$\phi \Rightarrow \phi \vee \phi$	(Apply $Rule_4$)
$\Rightarrow \neg\phi \vee \phi$	(Apply $Rule_2$,replace the first ϕ by $\neg\phi$)
$\Rightarrow \neg q \vee \phi$	(Apply $Rule_1$,replace ϕ by q)
$\Rightarrow \neg DWEL(v) \vee \phi$	(Replace q by atomic formula $DWEL(v)$)
$\Rightarrow \neg DWEL(v) \vee \phi \vee \phi$	(Apply $Rule_4$, replace ϕ by $\phi \vee \phi$)
$\Rightarrow \neg DWEL(v) \vee q \vee \phi$	(Apply $Rule_1$, replace the first ϕ by q)
$\Rightarrow \neg DWEL(v) \vee Solllage \vee \phi$	(Replace q by atomic formula $Solllage$)
$\Rightarrow \neg DWEL(v) \vee Solllage \vee \neg\phi$	(Apply $Rule_2$, replace ϕ by $\neg\phi$)
$\Rightarrow \neg DWEL(v) \vee Solllage \vee \neg q$	(Apply $Rule_1$, replace ϕ by q)
$\Rightarrow \neg DWEL(v) \vee Solllage \vee \neg Spitz$	(Replace q by atomic formula $Spitz$)

[1] BNF is a formal notation and it is commonly used to define the syntax of a language in computer science, for example, the syntax of a programming language. The meaning of the BNF notations '::=', '|' and '⇒' are listed on page 109. It is not the scope of this thesis to explain BNF in more detail. Interested readers are referred to the above cited literature.

Based on these expression rules, $\neg DWEL(v) \vee Solllage \rightarrow$ is not a wff. Since the logical connective $\rightarrow$ occurs in the formula, $Rule_5$ must be applied in the steps of composing this formula. In $Rule_5$, the left hand side and the right hand side of $\rightarrow$ must be replaced by using another expression rule. However, the right hand side of $\rightarrow$ in $\neg DWEL(v) \vee Solllage \rightarrow$ cannot be replaced anymore by any matched expression rule because it is an empty string. Therefore, this formula is not a wff[2]. In the OBLH tool, the syntactic rules are defined as a grammar specification. This specification is then converted to a grammar parser written in Java using the well-defined Java package called JavaCC [Sun09].

Some constructions of wff are standardized and considered as normal forms. One of these normal forms is **D**isjunctive **N**ormal **F**orm (DNF). A DNF is a disjunction of conjunctive clauses c. Each clause consists of atomic formulas or negations of atomic formulas which are connected by $\wedge$ and these clauses are connected by $\vee$. A wff ϕ is considered in DNF if it is built based on the rules as follows:

$c ::= q|(\neg q)|(c \wedge c)$
$\phi ::= c|(c \vee \phi)$

In propositional logic, a set of proof rules is defined for the purpose of reasoning. They describe how formulas should be manipulated in order to deduce the conclusion. The process of applying the rules is called natural deduction and the conclusions are said to be made syntactically. In OBLH, reasoning is achieved semantically, therefore, natural deductions will not be discussed in detail. Interested readers are referred to [Lee08, BL99, VOQ88].

5.1.2 Semantics

The semantic part of propositional logic defines the meaning of the connectives and formulas. Truth values are assigned to a formula in order to give a meaning to a sentence. The truth values in propositional logic are true (`T`) and false (`F`). The assignment of a truth value to a formula is called an evaluation of a formula. This means, in propositional logic, each formula can be evaluated to be true or false. The meaning of a formula or a connective is expressed in a truth table. It shows all the possible evaluations of a formula or a connective. For example, the meaning of the propositional connectives $\neg$, $\wedge$, $\vee$ and $\rightarrow$ are presented as truth tables in Table 5.1. Table 5.1 shows the truth tables of these four logical connectives. In the truth table of the logical connective implication $\rightarrow$, the first two columns express the possible evaluations of the atomic formulas ϕ and ψ, the third column expresses the results of the evaluations. Each line is an evaluation of the formula.

If a statement is expressed in the form of a logical implication (e.g. $\phi \rightarrow \psi$), then one can conclude that "Whenever ϕ is evaluated to be true, then ψ must also be true". In other words, this formula is evaluated to be true or correct, when such combinations of truth values occur as shown in the truth table. Furthermore,

[2]Except in this location and in the example in Figure 4.2 on page 36, a wff is simply called a formula in this thesis.

	ϕ	$\neg\phi$
1	T	F
2	F	T

	ϕ	ψ	$\phi \wedge \psi$
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	F

	ϕ	ψ	$\phi \vee \psi$
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

	ϕ	ψ	$\phi \rightarrow \psi$
1	T	T	T
2	T	F	F
3	F	T	T
4	F	F	T

Table 5.1: Truth tables of $\neg$, $\wedge$, $\vee$ and $\rightarrow$

the formula is also evaluated to be true independent of the truth value of ψ if ϕ is evaluated to be false. For instance, in Example 6 on page 17, $Requirement_{2.3.1}$ can be expressed by the logical formula $EL(v) \rightarrow Solllage$. Based on the definition of logical implication, this formula means whenever a point is locked as a route, an overlap or flank protection element of another route, then it must be in a proper position for the requested overlap. If this is not the case, then the requirement is not fulfilled. In other words, $Requirement_{2.3.1}$ is evaluated to be false. It also means that if the point is not locked, then the requirement is fulfilled independent of the position of this point. These evaluations of the formula are expressed in Table 5.2. It shows that the meaning of this formula is exactly the same as the meaning of the system requirement 2.3.1.

	$EL(v)$	$Solllage$	$EL(v) \rightarrow Solllage$
1	T	T	T
2	T	F	F
3	F	T	T
4	F	F	T

Table 5.2: Evaluations of $EL(v) \rightarrow Solllage$

The truth table of another logical formulation of $Requirement_{2.3.1}$ is also shown in Table 5.3.

	$EL(v)$	$Solllage$	$\neg Solllage$	$EL(v) \wedge \neg Solllage$	$\neg(EL(v) \wedge \neg Solllage)$
1	T	T	F	F	T
2	T	F	T	T	F
3	F	T	F	F	T
4	F	F	T	F	T

Table 5.3: Evaluations of $\neg(EL(v) \wedge \neg Solllage)$

5.1.3 Reasoning

In OBLH, checking the correctness of static conditions of the interlocking logic is based on two concepts of propositional reasoning. They are the concept of logical consequence and logical equivalence. The first concept is expressed mathematically as follows:

ϕ_1,ϕ_2, ...,$\phi_n \models \psi$

$(\phi_1, \phi_2, ..., \phi_n)$ is a set of formulas and this set is called the premise. ψ is another formula and it is called the conclusion. $\models$ is called semantic entailment. This expression is called a sequent. The sequent can be read as following, if formulas in the premise are all evaluated to be true, then check whether ψ is also evaluated to be true. If this is the case, the sequent is said to be valid and ψ is the logical consequence of the premise $(\phi_1, \phi_2 ..., \phi_n)$.

In a semantic reasoning of logical consequence of formulas, truth tables of $(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n)$ and ψ are compared. Whenever the result of the evaluations of $(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n)$ is true, then ψ must also evaluated to be true in its truth table. In other words, one check whether the formula $(\phi_1 \wedge \phi_2 \wedge ... \wedge \phi_n) \rightarrow \psi$ holds.

For example, consider the sequent $(EL(v), (EL(v) \rightarrow Solllage)) \models Solllage$. The truth table of the premise is shown in Table 5.4. To check the validity of this sequent, one must find the results of the evaluations of the premise $(EL(v) \rightarrow Solllage) \wedge EL(v)$ are true and then check the corresponding evaluation of the conclusion $Solllage$. If the evaluation of $Solllage$ is true, then the sequent is valid and $Solllage$ is the logical consequence of $(EL(v) \rightarrow Solllage) \wedge EL(v)$. In this example, there is only one evaluation in which the premise is evaluated to be true (line 1), the evaluation of $Solllage$ is true in this case. Therefore, the sequent is valid.

The truth table of the premise of another sequent $(\neg EL(v), (EL(v) \rightarrow Solllage))$ $\models Solllage$ is shown in Table 5.5. This truth table shows that this sequent is invalid because of the evaluation indicated in line 4. There are two evaluations in which the premise is evaluated to be true. They are line 3 and 4. In line 4, the evaluation of the conclusion $Solllage$ is false. It does not fulfill the defined relation between the premise and the conclusion. Therefore, this sequent is invalid and $Solllage$ is not the logical consequence of $(\neg EL(v), (EL(v) \rightarrow Solllage))$.

	$EL(v)$	$Solllage$	$EL(v) \rightarrow Solllage$	$(EL(v) \rightarrow Solllage) \wedge EL(v)$
1	T	T	T	T
2	T	F	F	F
3	F	T	T	F
4	F	F	T	F

Table 5.4: Checking the logical consequence of two formulas semantically

	$EL(v)$	$Solllage$	$\neg EL(v)$	$EL(v) \rightarrow Solllage$	$(EL(v) \rightarrow Solllage) \wedge \neg EL(v)$
1	T	T	F	T	F
2	T	F	F	F	F
3	F	T	T	T	T
4	F	F	T	T	T

Table 5.5: The sequent $(\neg EL(v), (EL(v) \rightarrow Solllage)) \models Solllage$ is not valid

	Natural language	Propositional logic	Decision table
System requirements	Admissible	T	Y
Checking conditions	Inadmissible	F	N
Result	Wrong specification	Invalid sequent	Wrong specification

Table 5.6: Verification of system requirements against checking conditions

	Natural language	Propositional logic	Decision table
Situation	Admissible	T	Y
System requirements	Inadmissible	F	N
Result	Inadmissible situation	Invalid sequent	Inadmissible situation

Table 5.7: Situational analysis of a situation against system requirements

The logical consequence of two formulas can also be checked by applying the proof rules of propositional logic. In this case, the semantic entailment symbol is replaced by the syntactic entailment $\vdash$.

It has been mentioned in Section 4.3.4 that it is important to check whether those situations admissible under system requirements, are also admissible under the specification of checking conditions for verification, like the expected behavior of the system or safety requirements. This checking is exactly the same as the definition of a sequent (see Table 5.6). Therefore, the concept of a sequent is applied in OBLH to verify the system requirements against the expected behavior or safety requirements. It is defined in Method 4. In other words, when system requirements are specified and a checking condition for verification are defined as propositional formulas SR and CC, respectively, then verification is modeled as a sequent $SR \models CC$. Since the formulas are expressed in the form of decision tables, the sequent in steps 3 of Method 4 is written as $DT_{SR} \models DT_{CC}$. It is shown in Tables 5.4 and 5.5 that those evaluations of the premise (system requirements) which cause the invalidity of the sequent can be found and therefore, those wrongly specified system requirements can be indicated and discovered.

The concept of checking the fulfillment of a situation S to the system requirements SR (see Section 4.3.2) can also be defined by the same concept $S \models SR$. In this sequent, one assumes that the situation is admissible and then it is important to check whether this situation is also admissible under the specification of the system requirements (see Table 5.7). If this sequent is invalid, then this means the situation is not admissible . This sequent can only be invalid if the situation is specified as inadmissible in the system requirements. Since the system requirements and the situation are expressed in the form of decision tables and the situation is also specified in a decision table, the sequent is written as $DT_S \models DT_{SR}$.

	Natural language	Propositional logic	Decision table
First requirement	Admissible	`T`	`Y`
Second requirement	Inadmissible	`F`	`N`
Result	Wrong specification	Invalid sequent	Wrong specification

(a)

	Natural language	Propositional logic	Decision table
First requirement	Inadmissible	`F`	`N`
Second requirement	Admissible	`T`	`Y`
Result	Wrong specification	Invalid sequent	Wrong specification

(b)

Table 5.8: Analysis of two system requirements (a) Case 1 (b) Case 2

By comparing between two system requirements, the inconsistent evaluation results of these two system requirements over the same situation must be found. The differences can be described by two cases (see Table 5.8). The first case is that the situation is evaluated to be admissible under the first requirement and is not evaluated to be admissible under the second requirement. In the second case, the situation is not evaluated to be admissible under the first requirement and is evaluated to be admissible under the second requirement. Therefore, in order to find out the inconsistent specifications, both cases need to be checked. As it has been shown above and in Table 5.8, one can formally define these checkings by using the concept of logical consequence. Given that the first requirement and the second requirement are specified as formulas $Requirement_1$ and $Requirement_2$, respectively, the checking of the first case is written as a sequent $Requirement_1 \models Requirement_2$ (see Step 3 of Method 1). The checking of the second case is written as another sequent $Requirement_2 \models Requirement_1$ (see Step 4 of Method 1). Since both cases must be analyzed, both step 3 and step 4 are defined in Method 1. The inconsistent specifications are indicated by the evaluations of the premise that cause the invalidity of the sequent (see Table 5.5).

Mathematically, logical equivalence of $Requirement_1$ and $Requirement_2$ are said to be checked in this case. If no difference in evaluations of two formulas is found, then these two formulas are said to be logically equivalent. Furthermore, if the validity of sequents are checked semantically, then these formulas are said to be semantically equivalent. In other words, they express the same meaning.

The truth tables in Table 5.2 and Table 5.3 are the propositional formalizations of the system requirement $Requirement_{2.3.1}$. Imagine that these two formulas (e.g. $Requirement_3$, $Requirement_4$) are specified in two different chapters in the interlocking system requirements specification and the corresponding written texts are not provided. They describe the same concept but are defined with different

logical connectives. Based on the concepts of OBLH, it is then important to check whether an inconsistency exists among these two specifications (formulas), this means logical equivalence of $Requirement_3$ and $Requirement_4$. First of all, the validity of sequent $Requirement_3 \models Requirement_4$ is checked. The results of evaluations of $Requirement_3$ which are true must also evaluated to be true in the truth table of $Requirement_4$. Secondly, the validity of sequent $Requirement_4 \models Requirement_3$ is checked. The results of evaluations of $Requirement_4$ which are true, must also evaluated to be true in the truth table of $Requirement_3$. No difference in evaluation is found in both checkings. Therefore, $Requirement_3$ and $Requirement_4$ define the same system requirement.

5.2 Introduction to Decision Tables

Decision tables are utilized in various concepts and functions of OBLH. For example, the process of verification and analysis of interlocking system requirements are carried out by specifying system requirements or safety requirements in decision tables. The results of checkings are also indicated in decision tables. Since it plays an important role in this formal framework, decision tables of OBLH and their properties must be formally defined.

In this section, decision tables are introduced both informally and formally in Section 5.2.1. The properties of decision tables and those that are important in OBLH are defined in Section 5.2.2.

5.2.1 Definition of Decision Tables

Decision tables are discussed informally based on the following example. Figure 5.1 is a decision table. This decision table specifies the system requirement 2.4 (ZPZ conditions for an infrastructure object located within the overlap and in German: ZPZ-Bedingungen je Fahrwegelement im Durchrutschweg) of the specification [Tut06]. The first column of the decision table is composed of two parts: the conditions of the system and the actions that are triggered based on the conditions. The first part is called condition stub and the second part is called action stub. The action stub contains the actions of the system. The condition stub contains all the possible conditions that will trigger the actions. In Figure 5.1, row 1 - 5 of the first column are the conditions of infrastructure elements and row 6 of the first column is the action. The action of this table is $ZPZ_{DWegElement}$.

The other columns show the possible combinations of the condition entries and the corresponding action entry. These columns are called the rules of a decision table. In this example, the condition and action entries are specified by either `Y` or `N`. They indicate Yes and No, respectively. The condition entry is described by these values. It is called a limited condition entry. The condition can be formulated as a yes-no question. For example, in Figure 5.1, the first condition is *Solllage*, this can be formulated as "Is the point in a proper position for the requested overlap?" If a condition is specified by different attributes, it is called extended condition entry. For example, a condition like "The aspects of a main

signal" can be Hp0, Ks1, Ks2. There is another symbol in the condition entry '–'. This entry means "do not care". This entry is called '*Don't-care*' entry. The action of this rule will not be affected by this condition. In other words, no matter whether this condition is evaluated to `Y` or `N`, it will lead to the same result in the corresponding rule.

In Figure 5.1, the action entry contains `Y` or `N` which indicates Yes and No respectively. `Y` means the action would be triggered by the combination of conditions. If the corresponding action will not take place, `N` is specified in the action entry. The third column (R2) indicates that the point is in an improper position for the requested overlap and it is a manually locking point, then $\mathrm{ZPZ}_{DWegElement}$ is negative. Similar to the condition entry, if the action entry is specified by the values, `Y` or `N`, it is called limited action entry and if it is specified by different attributes, it is called extended action entry.

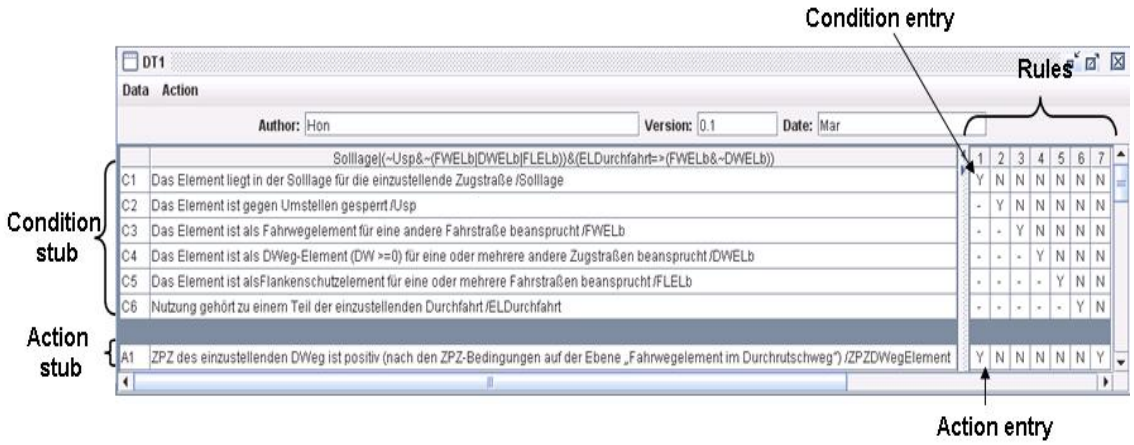


Figure 5.1: The evaluation of $\mathrm{ZPZ}_{DWegElement}$ in a decision table

In OBLH, propositional logic is used to specify the properties of infrastructure objects. It has been mentioned in Section 5.1.2 that the meaning of a proposition is given by the assignment of truth values. There are two truth values which are `T` and `F` and they are identical to `Y` and `N` respectively. As a result, decision tables with limited condition and action entries are used to express the static conditions. A part of definitions of a well-formed OBLH decision table DT_{OBLH} is given formally as follows:

- **Definition 1** *Condition set*
  $C = \{C_1, C_2, ..., C_c\}$

  The condition set indicates the set of condition symbols. In the railway domain, C contains all the possible attributes of infrastructure objects. For example,
  $\mathrm{C}_{ESTW-R} = \{Solllage, FWEL(b), DWEL(b), FLEL(b), BHSS, Spitz\}$.

- **Definition 2** *Condition domain*
  $CD = \{\{\mathtt{Y}, \mathtt{N}\}\}$

The condition domain CD indicates the possible values of the condition symbols. As mentioned above, propositions that are used to describe the properties of each infrastructure object has only two truth values. Therefore, the condition domain CD has only one subset which has only two elements, `Y` (Yes) and `N` (No).

- **Definition 3** *Condition subject*
  $CS = \{CS_1, CS_2, ..., CS_k\}$
  k: number of conditions in a decision table and $CS \subseteq C$

  The condition subject indicates a set of condition symbols of a decision table. This means, the condition subject must be a subset of the condition set. In Figure 5.1, the condition subjects are $Solllage$, Usp, $FWEL(b)$, $DWEL(b)$, $FLEL(b)$ and Bsp[3].
  $\mathrm{CS}_{ZPZ_{DwegElement}} = \{Solllage, Usp, FWEL(b), DWEL(b), FLEL(b), Bsp\}$.

- **Definition 4** *Condition*
  $C_i = (CS_{t_i}, \{\mathtt{Y}, \mathtt{N}\})$

  A condition is an ordered pair. Each pair indicates the condition subject and its possible values. If the possible values of a condition subject are `Y` and `N`, then the condition is called limited condition entry. In Figure 5.1, the condition Solllage is specified as an ordered pair $(Solllage, \{\mathtt{Y}, \mathtt{N}\})$.

- **Definition 5** *Condition space*
  $SPACE(C) = \{\mathtt{Y}, \mathtt{N}\}^k$

  Condition Space indicates the possible combination of the condition entries in a decision table. Since all condition entries are limited condition entries, the total number of possible combinations is $|SPACE(C)|$ is 2^k without the introduction of '*Don't-care*' entries. For example, the total number of conditions of the decision table in Figure 5.1 is 6 and the total number of possible combinations is then 64.

- **Definition 6** *Action set*
  $A = \{A_1, A_2, ..., A_a\}$

  The action set is a set of action symbols. It contains all the possible actions that could be triggered in the domain. For example, in the railway domain, $\mathrm{A} = \{ZPZ_{DWegElement}, ZPZ_{DWeg}\}$.

- **Definition 7** *Action domain*
  $AD = \{\{\mathtt{Y}, \mathtt{N}\}\}$

  Similar to the condition domain, it contains the possible values of each action symbol. There are only two possible values of the action symbol that are

[3]In Figure 5.1, the action subjects $FWEL(b)$, $DWEL(b)$ and $FLEL(b)$ are indicated without the pair of brackets. It is because brackets cannot be used to specify a proposition in OBLH tool. To remain the consistency of the definition of decision tables, $FWELb$ is considered as the same as $FWEL(b)$.

used in DT_{OBLH}. They are `Y` (Yes) and `N` (No). They indicate whether a safe route can be built during each phase of the development process under the logic of OBLH.

- **Definition 8** *Action subject*
  $AS = \{AS_k\}$
  $AS_k \in A$

  The action subject is a set of action symbols in a decision table. From this definition, each DT_{OBLH} can only have one action symbol and it has been defined in the action set. In Figure 5.1, the action subject is $\mathrm{ZPZ}_{DwegElement}$. $\mathrm{AS}_{ZPZ_{DwegElement}} = \{ZPZ_{DwegElement}\}$.

- **Definition 9** *Action*
  $A_i = (AS_{m_i}, \{\mathtt{Y}, \mathtt{N}\})$

  The action is a set of ordered pair of action symbols and their possible values. If the possible values of an action subject are the truth values `Y` and `N`, then the action is called limited action entry. In Figure 5.1, the Action is specified as an ordered pair $(ZPZ_{DwegElement}, \{\mathtt{Y}, \mathtt{N}\})$.

- **Definition 10** *Action space*
  $SPACE(A) = \{\mathtt{Y}, \mathtt{N}\}$

  The action space indicates the possible pairs of actions in the decision table. Each DT_{OBLH} can be specified by one action. As a result, the number of elements in this set is two.

- **Definition 11** *Decision table as matrix*

  $$DT = (dt_j) = \begin{pmatrix} d_{ij} \\ a_{1j} \end{pmatrix}$$

  $i \in \{1, ..., k\}$ *and* $j \in \{1, ..., n\}$
  k: number of conditions in the decision table
  n: number of rules in condition tables
  $d_{ij} \subseteq 2^{\{\mathtt{Y},\mathtt{N}\}} \setminus \{\emptyset\}$
  $a_{1j} \in \{\mathtt{Y}, \mathtt{N}\}$

  DT_{OBLH} is defined as a matrix. Informally speaking, dt_j represents a rule in a decision table. Each of the rules is composed of conditions and one action. There is one action which is defined in the action subject. As mentioned above, '*Don't-care*' entries can be used to combine rules together. '*Don't-care*' entries is represented as '–' and defined as d_{ij} = '–' which corresponds to $\mathrm{d}_{ij} = \{\mathtt{Y}, \mathtt{N}\}$.

5.2.2 Decision Tables in Object-Based Lastenheft

A part of the definitions of DT_{OBLH} has been given in the last section (from Definition 1 to Definition 11). Properties of decision tables are discussed in this

section. DT_{OBLH} must satisfy two of these properties which are consistency and completeness $DT_{Consistency}$ and $DT_{Completeness}$.

The first property of a decision table is called rule overlapping. Rule overlapping means that the intersection of all condition entries of two rules is not empty. Figure 5.2 shows the three possible rule overlappings of decision tables. In Figure 5.2(a), R1 and R2 are the same and the intersection of the condition entries is (Y, Y, N). The intersection of the condition entries is (Y, Y, N) in Figure 5.2(b). In Figure 5.2(c), the intersection is (Y,Y,–). Overlapping $DT_{Overlap}$ is formally defined as follows:

- **Definition 12** $DT_{Overlap}$
  $(\exists j,k \in \{1,...,n\})((\forall i \in \{1,...,c\})\ j \neq k \wedge (d_{ij} \cap d_{ik} \neq \emptyset))$

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	N	N	N

(a)

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	--	N	N

(b)

	R1	R2	∩(R1,R2)
X_1	--	Y	Y
X_2	Y	--	Y
X_3	--	--	--

(c)

Figure 5.2: Overlapping of rules in decision tables

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	N	N	N
a	Y	Y	

(a)

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	--	N	N
a	Y	N	

(b)

	R1	R2	∩(R1,R2)
X_1	--	Y	Y
X_2	Y	--	Y
X_3	--	--	--
a	Y	N	

(c)

Figure 5.3: Consistency of rules in decision tables

If a decision table contains overlapping rules, two aspects need to be considered and analyzed. They are consistency and exclusiveness. A decision table is called inconsistent if the action entries of any overlapping rules are different. In other words, a same situation can lead to different actions based on the decision table. There exists a contradiction in the decision table. In Figure 5.3(a), although the intersection of two rules is not empty, these rules are consistent because the actions of these rules are the same. Figure 5.3(b)

and (c) have inconsistencies among rules because there are overlapping rules, but their actions are different. Consistency $DT_{consistency}$ of decision tables is defined as follows:

- **Definition 13** $DT_{consistency}$
  $(\forall j,k \in \{1,...,n\})((\forall i \in \{1,...,c\})\ (d_{ij} \cap d_{ik} \neq \emptyset)) \rightarrow a_{1j} = a_{1k})$

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	N	N	N
a	Y	Y	

(a)

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	Y	Y
X_3	--	N	N
a	Y	Y	

(b)

	R1	R2	∩(R1,R2)
X_1	--	Y	Y
X_2	Y	--	Y
X_3	--	--	--
a	Y	Y	

(c)

	R1	R2	∩(R1,R2)
X_1	N	Y	∅
X_2	Y	--	Y
X_3	--	--	--
a	Y	Y	

(d)

Figure 5.4: Exclusiveness of rules in decision tables

If a decision table contains overlapping rules and the rules are consistent, exclusiveness can be considered. In other words, there is no ambiguity in choosing rules to be applied with a given condition pair. For example, in Figure 5.4(a), (b) and (c), it is not clear which rule should be applied with a given condition tuple (Y, Y, N) because of the overlapping. Figure 5.4(d) shows one of the possibilities in ensuring the exclusiveness of rules of the decision table in Figure 5.4(c). Exclusiveness $DT_{Exclusiveness}$ is formally defined as follows:

- **Definition 14** $DT_{Exclusiveness}$
  $(\forall j,k \in \{1,...,n\})((\exists i \in \{1,...,c\})\ (d_{ij} \cap d_{ik} = \emptyset))$

  Inclusiveness of a decision table means if two rules lead to the same action and contain only one difference in the condition entry, then these two rules can be combined and this condition entry of the rule must be represented by the union of the condition domain Y, N, in other words, the symbol '–'. In Figure 5.5, the intersection of two rules has two elements, as a result, Inclusiveness $DT_{Inclusiveness}$ is defined as follows:

- **Definition 15** $DT_{Inclusiveness}$
  $(\forall j,k \in \{1,...,n\})((\exists i \in \{1,...,c\})(\forall t \in \{1,...,c\}\text{-}\{i\})\ (d_{ij} \cap d_{ik} = \emptyset) \wedge$
  $(d_{tj} = d_{tk}) \wedge\ a_{1j} = a_{1k} \rightarrow d_{ij} = \{Y, N\})$

  Among the idea of rule overlapping and the properties of decision tables, there is another concept of decision tables that needs to be introduced. Given a decision table is consistent, it is said to be completed, if the possible combinations of condition entries are specified in the decision table. In Figure

	R1	R2	∩(R1,R2)
X_1	Y	Y	Y
X_2	Y	N	∅
X_3	N	N	N
a	Y	Y	

Figure 5.5: Inclusiveness of rules in decision tables

5.6, the decision table is not complete because there is one rule missing. This rule is (N, N, Y). The completeness of a decision table is formally defined as follows:

- **Definition 16** $DT_{Completeness}$
  $(\forall x \in SPACE(C))((\exists j \in \{1, ..., n\})((\forall i \in \{1, ..., k\})(x_i \in d_{ij})))$

	R1	R2	R3
X_1	Y	N	N
X_2	--	Y	N
X_3	--	--	N
a	Y	Y	N

Figure 5.6: An incomplete decision table

- **Definition 17** *A well-formed OBLH decision table* DT_{OBLH}
  A decision table is said to be well-formed in OBLH if it satisfies the definitions 1 to 11, the properties $DT_{Consistency}$ *and* $DT_{Completeness}$.

Those decision tables that are used to specify interlocking system requirements must satisfy this definition. The definitions 1 to 11 are automatically fulfilled if requirements are specified by using the OBLH tool. Furthermore, completeness and consistency of a decision table can be checked in the tool. The corresponding mathematical operations are defined in Section 5.6.1 and Section 5.6.2.

5.3 Propositional Logic and Decision Tables

To support analysis of meaning of a propositional formula, one can transform a formula into two forms: a truth table or a decision table in the OBLH tool. It has been mentioned in Section 5.1.2 that truth tables and propositional formulas are semantically equivalent. A complete and consistent decision table is also semantically equivalent to the corresponding propositional formula. Because of this property, mathematical structures can be mapped to one another. The mapping can be implemented as programs or techniques, such that it is carried out automatically. The suitable technique that is used to carry out the transformation, must also support the implementations of other concepts of OBLH and must be mathematically well-defined. The technique OBDD satisfies these requirements and is therefore used to implement the concepts of OBLH.

The relations between decision tables and propositional formulas are first given in Section 5.3.1. Using the technique OBDD as the basis to implement the concepts of OBLH is then discussed in Section 5.3.2.

5.3.1 Propositional formulas and Decision Tables

The equivalence relationship between propositional formulas and decision tables can be understood informally as follows: When a system requirement of the interlocking logic is specified in a decision table, like the decision table shown in Figure 5.1 on page 74, one first defines the conditions of objects called objects' attributes, that need to be checked in this requirement. The combinations of these conditions specify all possible situations that are under consideration w.r.t. the development of safe routes. In decision tables, the combinations are expressed by using values `Y` and `N`. Then, the evaluation result of each combination is specified in the action entry with values `Y` and `N`.

This way of specifying a system requirement in a decision table is the same as the evaluation of a formula in a truth table. A truth table consists of all propositions of the formula and their possible evaluations with truth values `T` and `F`. These two components are analog to the conditions of the decision table and the combinations of condition entries. The assignment of `T` and `F` to the result of each evaluation in propositional logic is then the same as the assignment of `Y` and `N` to each combination of conditions in a decision table. Their structural relationships are listed in Table 5.9.

Truth table	Decision table
Proposition	condition
Proposition formula	action
Interpretation(proposition)	Limited condition entry
Interpretation(formula)	Limited action entry
`T`	`Y`
`F`	`N`

Table 5.9: Equivalent structural relationships of truth tables and decision tables

Mathematically, the assignment of truth values in propositional logic is expressed as a function. This function maps each interpretation of the propositions to the set of truth values as follows:

$f : \{1, 0\}^n \rightarrow \{1, 0\}$
n: number of propositions in the formula

Such a function is called a boolean function. Propositions of the formula are called variables in the boolean function. Two truth values true `T` and false `F` are identical to `1` and `0` respectively. They are interchangeable in this work. A boolean function capture the semantic meaning of a propositional formula. Boolean functions are semantically equivalent to decision tables [ZB99]. Since propositional

formulas are also semantically equivalent to boolean functions[4], the same relation holds between decision tables and propositional formulas.

Propositional formula	Boolean function
Proposition (ϕ, ψ)	boolean variable (a, b)
T	1
F	0
$\neg\phi$	$\overline{a}$
$\phi \wedge \psi$	$a \cdot b$
$\phi \vee \psi$	$a + b$
$\phi \rightarrow \psi$	$\overline{a} + b$

Table 5.10: Equivalent structural relationships of propositional formulas and boolean functions

5.3.2 Propositional Formulas, Decision Tables and Ordered Binary Decision Diagrams

A decision table for specifying interlocking system requirements is said to be well-formed when it satisfies Definition 17. It must be complete and consistent. Therefore, the transformed decision table from a propositional formula must also satisfy these two properties. In order to facilitate the process of analyzing decision tables, the transformed decision table must also satisfy the property $DT_{Exclusiveness}$. Two rules with the same action entry have at least one difference among their condition entries. Furthermore, one of the advantages of using decision tables as a specification method is that the requirements can be expressed in a compact form in a decision table by combining rules. If there are rules with only one difference in a condition and the combinations of other conditions are the same and they lead to the same action, then this difference is represented by the '*Don't-care*' entry. These rules are then combined together and form a more compact decision table. This is called the optimization of decision tables. A compact decision table summarizes the possible situations that the system needs to react to and the corresponding actions. Finding the smallest size of a decision table, or in other words, the most optimized decision table, is an NP-hard problem [ZB99]. Therefore, the focus of this work has not been put on finding the most compact decision table for the corresponding rule. However, the transformation technique in the OBLH tool must also provide a mechanism to minimize the number of rules or in other words, remove the redundant rules.

Table 5.9 has shown that a truth table of a propositional formula can be transformed directly into a decision table [Mon74]. Through such a transformation, decision tables contain all the combinations of condition entries and the corresponding actions without introducing '*Don't-care*' entries. In this case, $DT_{Consistency}$,

[4]In a finite set, boolean expressions are isomorphic to propositional logic, informally, this means propositional algebra and boolean expressions can be described as equivalent [Joh93]. Their structural relationships are listed in Table 5.10.

$DT_{Completeness}$ and $DT_{Exclusiveness}$ can be guaranteed based on this transformation. However, minimizing the number of rules is not supported. Furthermore, one of the requirements of the transformation technique is that it must also support mathematical operations, such that decision tables can be analyzed. No corresponding mathematical operations can be applied if this way of transformation is used. Algorithms need to be developed additionally if this transformation technique is used.

OBDDs are the type of technique [Bry86] (see Section 5.4) that fulfill all the mentioned requirements of a transformation technique. An OBDD with the best variable ordering is a compact form to express a boolean function or in other words a decision table. OBDDs are based on the concept of binary decision trees and reduction rules are applied to the trees to obtain a compact representation of boolean functions. Applying these reduction rules to the binary decision trees results in a decision table with the corresponding '*Don't-care*' entries and the number of rules in the decision table is reduced (see Section 5.4.2). There are different advantages of applying this technique for transformation (see Figure 5.10). First, a decision table that is generated based on the definition of OBDD fulfills $DT_{Consistency}$, $DT_{Completeness}$ and $DT_{Exclusiveness}$. If a formula or a decision table is contradictory, only the zero terminal will be generated in an OBDD. Secondly, mathematically well-defined OBLH concepts can be implemented by applying the defined boolean operations of OBDDs because of the equivalent relationship. The defined concepts in the form of propositional formulas can be mapped to boolean functions. The corresponding propositional operations can be achieved by applying boolean operations. Since the OBLH concepts are defined by propositional logic, boolean operations provided by the OBDD technique can then be used to implement these concepts, for example, checking logical consequence of system requirements. The OBDD package is a well-developed program that generates OBDDs based on boolean functions and supports manipulation of OBDDs with boolean operations [BRB90]. This package can then be used as the underlying implementation of the OBLH tool. The OBDD package that is used in the OBLH tool is JADE [Dre02].

The mechanism for minimizing the number of rules in decision tables depends on variable ordering of OBDDs. One of the drawbacks of this technique is finding the best variable ordering of OBDDs. This ordering affects the application of the reduction rules to OBDDs. And therefore, it affects the size of an OBDD representing a boolean function and the corresponding decision table [BW96, ZB99]. In Figure 5.7(a), the ordering of the variables is ($DWEL(b)$, $Solllage$, $Spitz$, $BHSS$), while in Figure 5.7(b), it is ($Spitz$, $BHSS$, $Solllage$, $DWEL(b)$). The first OBDD is smaller than the latter one. This implies, the number of rules in the decision table will be affected (see Figures 5.8 and 5.9). As it has been mentioned above, the focus of this work is not to generate the smallest decision table, therefore an investigation of the best ordering for a boolean function is not done. However, the setting of the suitable initial variable ordering for generating an OBDD has been considered based on the characteristics of the reduction rules. In Section 5.5.1, the initial variable ordering of an OBDD representing the boolean function (or the

propositional formula) is discussed. Figure 5.10 describes the activities and the decisions that have to be made during the transformation of a propositional formula to a decision table with OBDD. This algorithm is called $AOBDD_{Transformation}$.

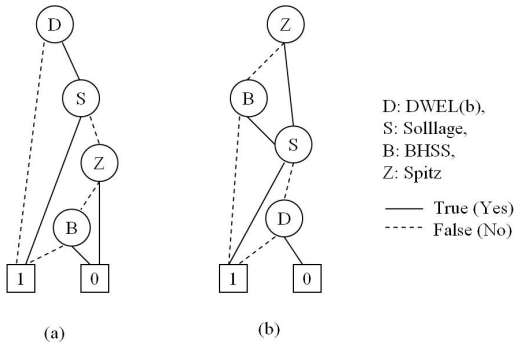


Figure 5.7: OBDDs of $Requirement_{1.7}$

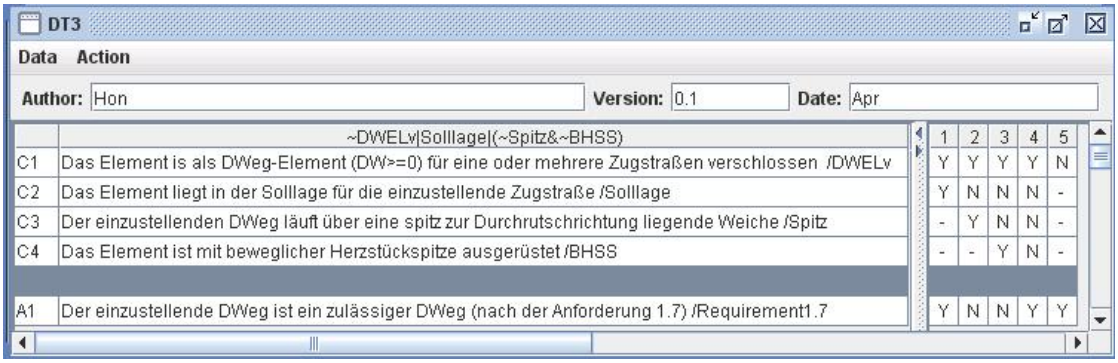


Figure 5.8: A decision table of the OBDD in Figure 5.7(a)

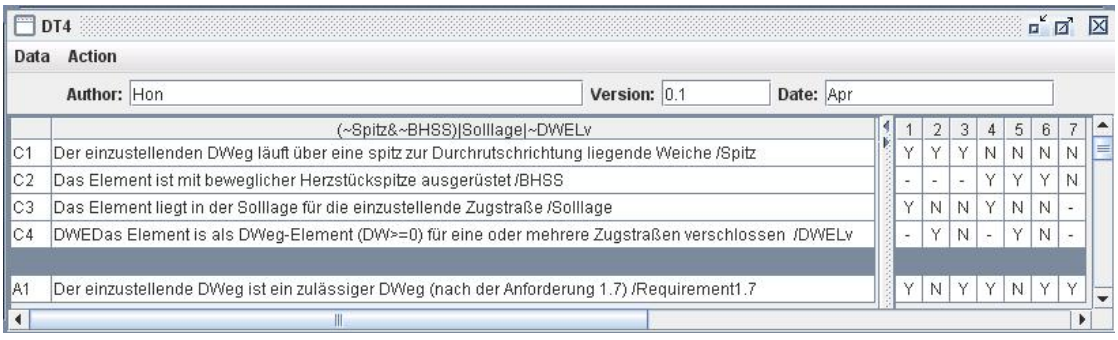


Figure 5.9: A decision table of the OBDD in Figure 5.7(b)

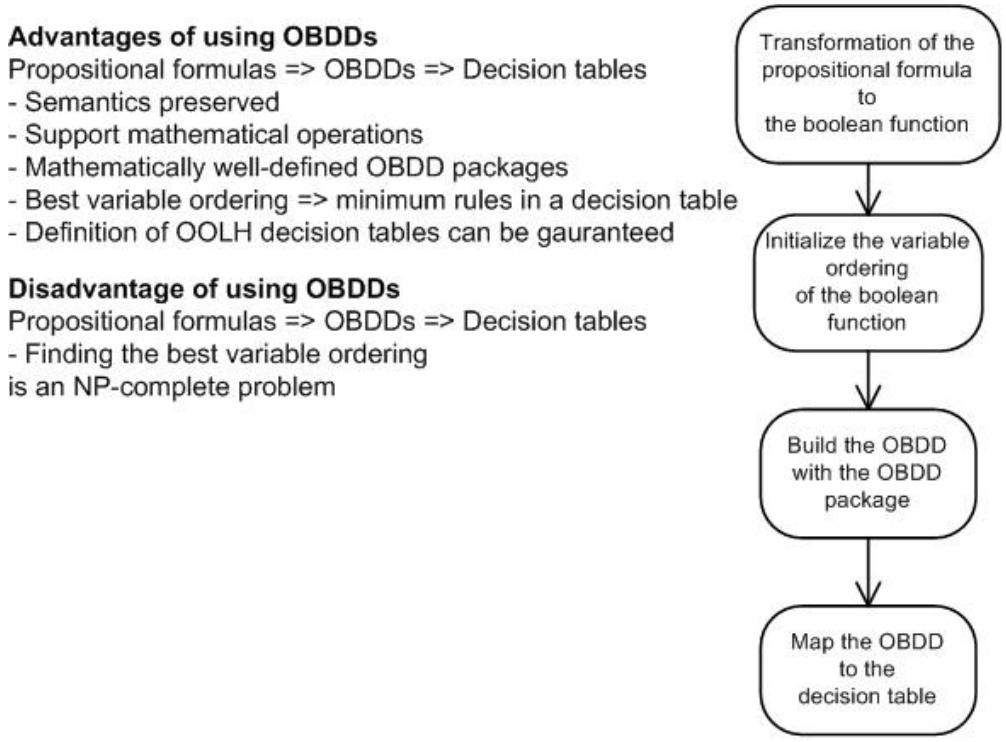


Figure 5.10: The transformation from propositional formulas to decision tables with OBDDs

5.4 Introduction to Ordered Binary Decision Diagrams

The advantages of using OBDDs as the transformation technique in this work has been briefly discussed in the above section. In this section, an introduction to OBDDs is first given. OBDDs are defined based on three different concepts. These concepts are **B**inary **D**ecision **D**iagram (BDD), **R**educed **B**inary **D**ecision **D**iagram (RBDD) and variable orderings. The first concept is discussed in Section 5.4.1. An introduction to RBDDs is given in 5.4.2. The concept of variable orderings and OBDDs is elaborated in Section 5.4.3.

5.4.1 Definition of Binary Decision Diagrams

A boolean function can be represented by a truth table. A truth table can be viewed as a decision table without the properties of $DT_{Inclusiveness}$. This means the directly transformed decision table from a boolean function contains redundant rules (see Section 5.3.2). A boolean function can also be represented by a BDD (see Figure 5.12 on page 87). If this binary decision diagram is directly transformed as a decision table, then this decision table fulfills Definition 17 and $DT_{Exclusiveness}$. However, redundant rules will exist. Before giving the formal definition of a BDD, some concepts and notations must be defined.

- **Definition 18** *Directed graphs*
  $G = (V, E)$
  V is a finite set of vertices and E is a finite set of edges

  A directed graph is defined by a finite set of vertices and a finite set of edges. In Figure 5.11, there are four vertices and five edges. It can be defined as a directed graph G as follows: $G = (V, E), V = \{v_1, v_2, v_3, v_4\}$ and $E = \{e_1, e_2, e_3, e_4, e_5\}$.

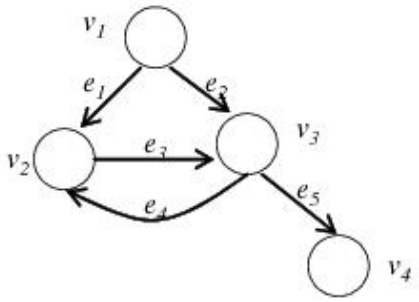


Figure 5.11: A directed graph

- **Definition 19** *Initial node and terminal node of an edge e*
  $initial : E \to V$
  $terminal : E \to V$

  An edge can be considered as a connection between two vertices. One of the vertices is the beginning of a connection, called the initial node of an edge e. The end of the connection is called the terminal node of the edge e. The function *initial* is defined to map the edge to its initial node and *terminal* maps the edge to its terminal node. In Figure 5.11, if $init(e_1) = v_1$ and $terminal(e_1) = v_2$, then e_1 is the edge that connects the vertices v_1 and v_2.

- **Definition 20** *A Path in a graph is a sequence of edges $e_1, e_2..., e_n$*
  $(\forall i \in \{1, ..., n-1\})(terminal(e_i) = initial(e_{i+1}))$
  $e_i \in E$

  A path is composed of at least one edge and two vertices. The edges of a path have the relationship that the terminal node of an edge is the initial node of the following edge. In Figure 5.11, a path from the vertices v_2 to v_4 is (e_3, e_5). The sequence e_3, e_5 forms a path because $initial(e_5) = terminal(e_3)$.

- **Definition 21** *A cycle in a graph is a path $e_1, e_2...e_n$*
  $terminal(e_n) = initial(e_1)$

  As a cycle, the terminal node of the last edge of a path is the initial node of the first edge of a path. In Figure 5.11, the edges e_3 and e_4 form a cycle. It is because $initial(e_4) = terminal(e_3)$. This sequence also forms a cycle because $terminal(e_4) = initial(e_3)$. A graph without a cycle is called an acyclic graph.

- **Definition 22** *$v_0 \in V$ is an initial node of a graph*
  , iff
  $(\forall e \in E)(terminal(e) \neq v_0)$

  An initial node v_0 of a graph is a vertex that does not have an incoming edge. In other words, the terminal node of any edge cannot be the initial node of the graph. In Figure 5.11, v_1 is the only initial node.

- **Definition 23** *$v \in V$ is a terminal node of a graph*
  , iff
  $(\forall e \in E)(initial(e) \neq v)$

  V_t is the set of the terminal nodes of a graph. A terminal node of a graph is a vertex that does not have an outgoing edge. In other words, the initial node of any edge of the graph cannot be the terminal node. The set of terminal nodes of Figure 5.11 is $\{v_4\}$.

- **Definition 24** *$v \in V$ is a non-terminal node of a graph*
  , iff
  $(\exists e \in E)(initial(e) = v)$

  A non-terminal node of a graph is a vertex that must have an outgoing edge. In other words, if a vertex v is a non-terminal node, there must exist an edge whose initial node is v. The set of non-terminal nodes of Figure 5.11 is $\{v_1, v_2, v_3\}$.

Definition 25 *BDDs*
A BDD has the following properties:

- *a directed acyclic graph*

- *an unique initial node $v_{initial}$*

- *non-terminal nodes are labeled with a function variable $: (V - V_t) \rightarrow L$ where L is a set of variables of a boolean function. Each non-terminal node has exactly two children that are assigned by two functions $low : (V - V_t) \rightarrow V$ and $high : (V - V_t) \rightarrow V$ and the properties:*

  $(\forall v \in (V - V_t))(\exists e_1, e_2 \in E)((initial(e_1) = initial(e_2) = v) \wedge (terminal(e_1) = low(v)) \wedge (terminal(e_2) = high(v)) \wedge (edge(e_1) = 0) \wedge (edge(e_2) = 1))$

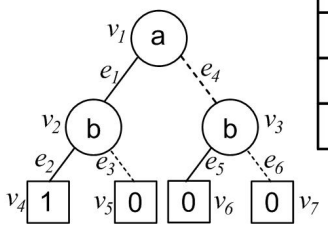
  $(\forall v \in (V - V_t))(|\{e \in E | initial(e) = v\}| = 2)$

  where $edge : E \rightarrow \{0, 1\}$

- *terminal nodes are labeled with a function value $: V_t \rightarrow \{0, 1\}$*

The usage of graphical symbols of BDDs is based on the paper [Bry86]. Figure 5.12 shows a BDD representation of the formula $a \wedge b$. Terminal nodes and

non-terminal nodes are represented by squares and circles, respectively. The initial node of the diagram is always drawn at the top. Edges labeled with 0 are represented by dashed lines, while edges labeled with 1 are drawn as solid lines. Furthermore, the direction of the edges will not be indicated in the graph. Normally, the orientation of an edge is defined from top to bottom.



	a	b	a ∧ b
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	F

$f_{a \wedge b}: \{1,0\}^2 \rightarrow \{1,0\}$

Figure 5.12: A BDD of $a \wedge b$, truth table and boolean function

The BDD in Figure 5.12 is formally defined as follows:
$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$
$E = \{e_1, e_2, e_3, e_4, e_5, e_6\}$
$L = \{a, b\}$
$V_t = \{v_4, v_5, v_6, v_7\}$
$v_{initial} = v_1$
$variable = \{(v_1, a), (v_2, b)\}$
$low = \{(v_1, v_3), (v_2, v_5), (v_3, v_7)\}$
$high = \{(v_1, v_2), (v_2, v_4), (v_3, v_6)\}$
$value = \{(v_4, 1), (v_5, 0), (v_6, 0), (v_7, 0)\}$
$edge = \{(e_1, 1), (e_2, 1), (e_3, 0), (e_4, 0), (e_5, 1), (e_6, 0)\}$

The vertices are defined without labeling in a directed graph. In BDDs, the non-terminal and terminal nodes are labeled with variables of the boolean function and values 0 or 1 respectively. Terminal nodes are the function values of a boolean function. Each of the non-terminal nodes has exactly two children. They are the low child and high child. They can be reached by traversing via the dashed line and solid line respectively. A BDD can be used to represent a boolean function. In Figure 5.12, the path from v_1 to v_4 represents (1,1,1) or the interpretation (T,T,T) in the truth table of this formula. One can read this path as if the variable a is assigned by the truth value T (1) and the variable b is assigned by the truth value T (1), then the truth value of this formula is T (1).

5.4.2 Definition of Reduced Binary Decision Diagrams

The BDD in Figure 5.12 has a redundancy. It can also be viewed as a redundance in a decision process. For example, when the value zero is assigned to the variable

a, the function value is 0. The function value is independent of the assignment of the variable b. The size of this BDD can be reduced if this redundance is removed. A BDD is a RBDD if the redundances in the BDD are eliminated based on three requirements of a RBDD [Bry86], [HR04] and [And98]. These three requirements are also considered as the reduction rules that apply to BDDs. These three requirements remove the unnecessary decision points in a BDD and reduce the size of a BDD. It is the same as consolidating rules in a decision table. These three requirements are defined in the following definition.

Definition 26 *RBDDs*
A RBDD is a BDD with the following properties:

1. *There exists only two terminal nodes labeled with 0 and 1.*
   $|V_t| = 2$ *and value is a bijective function.*

2. *No non-terminal node has the same children*
   $(\forall v \in (V - V_t))(\neg(low(v) = high(v)))$

3. *No non-terminal nodes with the same variable labeling have the same children*
   $(\forall v_1, v_2 \in (V - V_t))((variable(v_1) = variable(v_2) \wedge \neg(v_1 = v_2) \rightarrow \neg(low(v_1) = low(v_2) \wedge high(v_1) = high(v_2))))$

In a BDD, the non-terminal nodes can only be labelled 0 or 1, the size of a BDD can be reduced by sharing the non-terminal nodes. This idea leads to the first requirement of a RBDD. The second requirement does not allow the two outgoing edges of a node v_i point to the same node v_j. This means, the evaluation of the boolean function does not depend on the assignment of v_i. In this case, the vertex v_i will be removed and the incoming edges of v_i will be redirected to v_j. In Figure 5.13(a), one of the vertices with a label b points to the same terminal node 0. This vertex is then removed and its incoming edge is redirected to the terminal node 0 in Figure 5.13(b). The third requirement states that if there exists two nodes v_i and v_j with the same variable labeling sharing the same children or subgraph, then one of the nodes, say v_j and its outgoing edges must be removed. The incoming edges of v_j are then redirected to v_i. In Figure 5.14(a), the vertices v_2 and v_3 do not fulfil the last requirement of RBDD because variable(v_2) = variable(v_3), low(v_2) = low(v_3) and high(v_2) = high(v_3). As a result, one of the nodes must be removed. In 5.14(b), v_3 is removed. However, this graph is not an RBDD because low(v_1) = high(v_1). Based on the second requirement, v_1 is redundant and therefore it must be eliminated. Figure 5.14(c) is the valid RBDD.

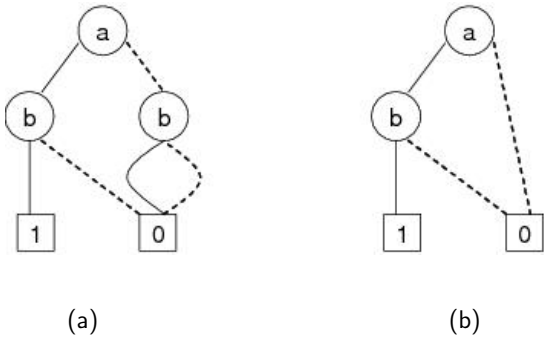


Figure 5.13: Reductions on a BDD of $a \wedge b$

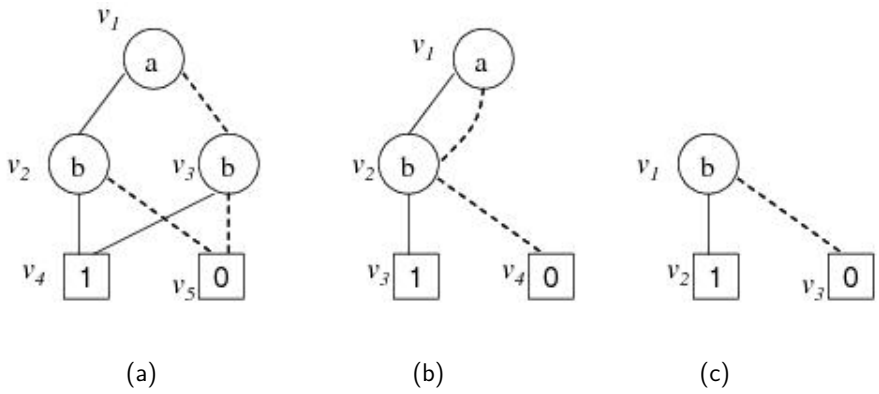


Figure 5.14: Reductions on a BDD

5.4.3 Definition of Ordered Binary Decision Diagrams

Figure 5.15 shows a valid RBDD, it satisfies the three requirements of RBDD. However, the path v_1, v_2, v_3 and v_4 is redundant because the sub-path from v_3 to v_4 will never be taken because $variable(v_1) = variable(v_3)$ and $edge(e_1) = 1$. This means the variable a has been assigned with the value 1 and cannot be assigned with a new value. The latter assignment will not be considered. Based on this concept, variable ordering is introduced to RBDD. A RBDD with variable ordering is called **O**rdered Reduced **B**inary **D**ecision **D**iagram (OBDD). It is formally defined as follows:

Definition 27 *OBDDs*
An OBDD is a RBDD with the following properties:
$(\forall e \in E)(terminal(e) \notin V_t \rightarrow (order(variable(initial(e)))$
$< order(variable(terminal(e)))))$ *order* $: L \rightarrow \{0, 1, ...n\}$ *and order is a bijective function*

In this work, if $order(variable(v_i)) < order(variable(v_j))$, then $variable(v_i)$ is said to be ordered higher than $variable(v_j)$. A path from the initial node to the last non-terminal node of an OBDD represents a subset of the domain of the boolean

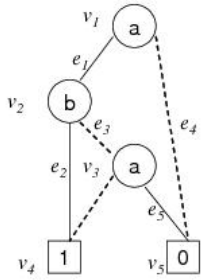


Figure 5.15: A BDD of $a \wedge b$ where a variable a occurs more than once

function. The terminal node of this path represents the corresponding projection in the co-domain. An OBDD *obdd* is used to evaluate the corresponding boolean function f and $transform : L \rightarrow \{1, 0\}$ in the following way:

Algorithm 1 $AOBDD_{Evaluation}$

1. *Start at* $v_{initial}$ *of obdd*
2. *while (current vertex* $v \notin V_t$*)*
   $\{$ *if* $(transform(variable(v)) == 0)$ $v = low(v)$; *else* $v = high(v)$; $\}$
3. *if* $(value(v) == 0)$ $f(x_1, x_2, ..., x_n) = 0$; *else* $f(x_1, x_2, ..., x_n) = 1$;

OBDD is the canonical representation of a boolean function [Bry86]. This property brings advantages in manipulating OBDDs and analyzing of boolean functions. First, redundant variables of a boolean function can be found by establishing the corresponding OBDD. Checking the equivalence of two boolean functions can be achieved by comparing the structure of the corresponding OBDDs. These OBDDs must have a compatible variable ordering. Furthermore, the validity of a propositional formula can be tested by establishing the corresponding OBDD. If this OBDD has only the node v, $v \in V_t$ and $value(v) = 0$, then the propositional formula is not valid. Finally, there is a set of defined boolean operations that can be applied to manipulate OBDDs. These boolean operations include $\overline{f}$, $f + g$ and $f \cdot g$.

The size of an OBDD in representing a boolean function depends on its variable ordering. In other words, OBDDs that are built based on the same boolean function with different variable orderings have different sizes. When an OBDD is used to represent a decision table, the number of rules is also affected by the size of the OBDD (see Figure 5.7, Figure 5.8 and Figure 5.9). As a result, finding the variable orderings for the boolean function is an important issue in using OBDDs. Static and dynamic approaches have been developed in finding the optimal ordering [BM02]. Strategies are designed to generate the ordering of an OBDD based on information of the specific application area in the static approach [FFK88]. Instead of using specific information, the variable ordering of a built OBDD is changed progressively in dynamic approaches. One of these strategies is the sifting algorithm. The main concept of this approach is swapping adjacent variables

locally to search for the best ordering. A variable is chosen to exchange the position with its adjacent variable upwardly and downwardly w.r.t. the OBDD, the minimum size of the OBDD and the corresponding position is then recorded. If the size of the current order is bigger than the expected, the process of swapping terminates. The algorithm continues to swap the next variable until all the variables have been used for estimation [Rud93] and [The97]. It is an NP-complete problem to find the best ordering to obtain the most compact OBDD [BW96]. Finding the smallest size of decision tables is an NP-hard problem [ZB99]. As mentioned above, the focus of this work is not to optimize the number of rules of a decision table. However, based on the provided reduction rules (see Definition 26), the setting of the initial variable ordering to generate an OBDD that represents the corresponding boolean function (or the propositional formula) has been considered. This is discussed in the next section.

5.5 Transformation of Ordered Binary Decision Diagrams and Decision Tables

In Figure 5.10, the process of transforming a propositional formula to a decision table is generally described. In this section, the idea is further elaborated. The algorithm is first discussed in Section 5.5.1. The idea of mapping an OBDD to a decision table is considered in Section 5.5.2.

5.5.1 Propositional Formulas and Ordered Binary Decision Diagrams

Algorithm 2 $AOBDD_{Transformation}$

1. $ordering = initial(formula)$;
2. $obdd = build(obdd, ordering)$;
3. $DT = transformToDT(obdd)$;

The first step of $AOBDD_{Transformation}$ is to transform the propositional formula to a boolean function and initialize the variable ordering of the OBDD. The propositional formula needs to be first transformed to a boolean function. It is because the OBDD package generates OBDDs based on boolean functions. A DNF (see Section 5.1.1 on page 67) of a propositional formula is equivalent to a boolean function in term of a sum of products (see Table 5.10). Therefore, the propositional formula is first transformed into DNF. As it has been mentioned before, the variable ordering of an OBDD affects the application of the reduction rules, or in other words, the number of rules in the decision table will be affected. In OBLH, the initialization of the variable ordering is based on the order of the propositions in the formula. The formulas that need to be transformed are static conditions of the interlocking system requirements and is defined by objects'

attributes. Each attribute has a meaning and therefore users can rank the importance of these attributes based on the meaning of the system requirements (KO criteria). The more important an attribute is, the higher its position in the written propositional formula. In order words, in the variable ordering of the OBDD, this variable is ordered higher than other variables, for example, $Requirement_{1.7} \equiv \neg DWEL(b) \vee Solllage \vee (\neg Spitz \wedge \neg BHSS)$. Based on the domain knowledge and the semantic meaning of logical connectives of the formula, the importance of these attributes is ranked as (DWEL(b), Solllage, Spitz, BHSS). The input formula to the OBLH tool is then $(\sim DWELb|Solllage|(\sim Spitz\& \sim BHSS))$. It has been shown in Figure 5.8 that if this rank is used as the variable ordering of the OBDD, then the corresponding decision table has only 5 rules. However, if one uses another ranking like the one in 5.7(b) $((\sim Spitz\& \sim BHSS)|Solllage| \sim DWELb)$, then the number of rules is increased. Therefore, the initial variable ordering of the OBDD is obtained based on the choice of users. The transformation of the propositional formula to a DNF and initialization of the variable ordering are implemented in the method $initial(formula)$.

In step 2 of $AOBDD_{Transformation}$, the initial variable ordering is used to built the OBDD via the OBDD package in the method $build(obdd, ordering)$ [Dre02]. After the OBDD is built by the OBDD package, it is mapped to a decision table based on an algorithm $AOBDD_{DT}$ described in the following section.

5.5.2 Ordered Binary Decision Diagrams and Decision Tables

The algorithm for evaluating a boolean function with the corresponding OBDD has been discussed in Section 5.4.3 (see Algorithm 1). A path of an OBDD represents the relation between the assignment of the boolean variables and the function values. In other words, each path of an OBDD is a representation of a rule of the decision table (see Table 5.11). Given a path $e_1, e_2, ..., e_m$ of an OBDD, it is mapped to a rule dt_j of the decision table DT as follows:

Algorithm 3 $AOBDD_{DT}$

1. $i = 1$; *while* ($i \leq m$)
   { $l = order(variable(initial(e_i)))$;
   if$(edge(e_i) == 0)$ $d_{lj} =$ `N`; *else* $d_{lj} =$ `Y`;
   if $(i == m)$*{ if* $(value(terminal(e_i)) = 0)$ $a_{1j} =$ `N`; *else* $a_{1j} =$ `Y`; *}*i_{++}; }

2. *while* $(i \leq k)$ {*if* (d_{ij} *is empty*) $d_{ij} =$ -; i_{++};}

$AOBDD_{DT}$ is implemented as the method $transformToDT(obdd)$ in $AOBDD_{Transformation}$. One of the assumptions of this algorithm is that the variable ordering of the function is the same as it is shown in the decision table. The set of condition subjects is ordered and the ordering is the same as the variable ordering of the OBDD. $(\forall l \in L)(\forall c \in CS)((l = c) \rightarrow (order(l) = position(c)))$, where $position : CS \rightarrow \{1, ..., n\}$. In Figure 5.16, one of the paths is e_1, e_2. Based

on the above algorithm, the result of step one is dt_2 = (Y, Y,,, Y) and the output of the algorithm is dt_2 = (Y, Y,-,-, Y) as it is shown as rule R1 in the decision table of Figure 5.8 on page 83.

OBDD	Decision table
Path	rule
L	CS
$edge(e) = 0 \wedge initial(e) = CS_i$	$dt_{ij} =$ N
$edge(e) = 1 \wedge initial(e) = CS_i$	$dt_{ij} =$ Y
$value(v) = 0 \wedge v \in (V - V_t)$	$a_{ij} =$ N
$value(v) = 1 \wedge v \in (V - V_t)$	$a_{ij} =$ Y
$\phi \rightarrow \psi$	$\overline{a} + b$

Table 5.11: The transformation of OBDDs and decision tables

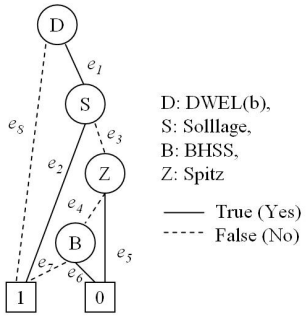


Figure 5.16: An OBDD of $Requirement_{1.7}$

As it has been mentioned above, the decision table that is obtained from an OBDD satisfies the requirements of DT_{OBLH} and $DT_{Exclusiveness}$. If the transformed decision table does not contain any overlapping rules, then these requirements are satisfied. As it has been defined in Section 5.2, in a decision table, if the intersection of a condition entry of any two rules is not empty, then this decision table does not contain any overlapping rules. Based on the mapping of OBDDs, no overlapping rules will be produced because of the property of OBDDs. First, duplicated variables along any path of OBDDs are not allowed. A path of an OBDD is mapped to a rule of the decision table, the last node of the path is a non-terminal vertex. A non-terminal vertex of an OBDD (corresponding to a condition) must have two out-going edges e_1 and e_2 (corresponding to the condition entries) and $edge(e_1) \neq edge(e_2)$. In other words, there exists at least one condition entry for any two rules that have no intersection.

A propositional formula is used to describe a true sentence or admitted situation(s). The situation(s) is(are) expressed by atomic formulas and the connectives in propositional logic. Similarly, in a DT_{OBLH}, each rule is used to describe a situation with the condition entries and the corresponding action entry. To transform a decision table to a propositional formula, rules with action entry Y are combined

by the connective $\vee$. The condition entries of each rule are connected by the connective $\wedge$. This propositional formula describes the logic of the decision table. As a result, based on the definition of decision tables in Section 5.2.1, the logic that is represented by a completed decision table DT_{OBLH} can be expressed by a propositional formula in DNF. Given a decision table DT, a DNF f is constructed as follows:

Algorithm 4 *$AOBDD_{DTDNF}$*

1. *find the rules with $a_{1j} = \mathtt{Y}$.*
2. *for each rule in this set of rules* {
   if ($d_{1j} == \mathtt{N}$) $c_m = \neg C_1$; else if ($d_{1j} = \mathtt{Y}$) $c_m = C_1$;
   else $\{c_m = \neg C_1;\ c'_m = C_1;\ add(0, c_m, P_j);\ add(|P_j|, c'_m, P_j);\}$
   $i = 1$;
   while ($i! = k$){
   if ($d_{ij} == \mathtt{N}$) $\{s = |P_j|;\ for(\ x = 0; x < s; x_{++}\ )\{c_x = c_x \wedge \neg C_i;$
   $add(x, c_x, P_j);\}\}$
   else if ($d_{ij} == \mathtt{Y}$) $\{s = |P_j|;\ for(\ x = 0; x < s; x_{++}\ )\{c_x = c_x \wedge C_i;$
   $add(x, c_x, P_j);\}\}$
   else $\{s = |P_j|;\ for(\ x = 0; x < s; x_{++}\ )\{c_t = c_x;\ c_x = c_t \wedge C_i;$
   $c'_x = c_t \wedge \neg C_i;\ add(x, c_x, P_j);\ add(|P_j|, c'_x, P_j);\}\}\}$
   }
3. $f = \bigvee_{j=1}^{n} \bigvee_{k=1}^{|P_j|} c_k$

where $P_j = \{\mathtt{Y}, \mathtt{N}\}^*$,
$c_g \in P_j, g \in \{0, 1\}$ *and*
$add : \mathbb{N} \times \{\mathtt{Y}, \mathtt{N}\} \times \{\mathtt{Y}, \mathtt{N}\}^* \rightarrow \{\mathtt{Y}, \mathtt{N}\}^*$

For example, the propositional formula that expresses the logic of the decision table in Figure 5.1 on page 74 is $(Solllage) \vee (\neg Solllage \wedge \neg Usp \wedge \neg FWELb \wedge \neg DWEL(b) \wedge \neg FLEL(b) \wedge \neg ELDurchfahrt)$. DNF can be used to build the corresponding OBDD.

5.6 Object-Based Lastenheft Concepts and Ordered Binary Decision Diagrams

The OBDD technique provides boolean operations for manipulating OBDDs. Since boolean functions, propositional formulas and decision tables are semantically equivalent, their operations are then interchangeable. The concepts of situational analysis, verification and analysis of interlocking system requirements are defined formally in propositional logic in Section 5.1.3. Propositional operations are defined to manipulate the specified situations, checking conditions for verification and system requirements, such that the goal of each concept can be achieved, for example, the fulfillment of a situation w.r.t. the system requirements can be checked. These propositional operations can be completed by the corresponding mapped boolean operations.

In the OBLH tool, the situations, checking conditions (e.g. safety requirements) and system requirements are specified in decision tables. These decision tables are transformed into OBDDs (see Algorithm 4) and defined propositional operations of each OBLH concepts are achieved by applying the corresponding boolean functions to these transformed OBDDs. As it has been mentioned before, the operations on OBDDs are accomplished by the well-developed OBDD package, therefore, no extra effort needs to be put in developing a program that executes boolean operations. The correctness of the results is then guaranteed because of the formal definition of OBDDs. For each OBLH concept, algorithms are designed to describe the process of generating the OBDDs from decision tables, applying boolean operations and interpreting the results of the operations.

These algorithms are discussed in this section. Checking the consistency and completeness of a decision table is discussed in Sections 5.6.1 and 5.6.2, respectively. Implementation of situational analysis and analysis and verification of system requirements with OBDDs is then elaborated in the final section.

5.6.1 Checking Consistency of Decision Tables

Given a decision table DT, the consistency of rules in DT is checked based on the following steps:

Algorithm 5 $AOBDD_{Consistency}$

1. *Build* $obdd_Y$ *based on the rules with* $a_{1j} = \mathtt{Y}$
2. *Build* $obdd_N$ *based on the rules with* $a_{1j} = \mathtt{N}$
3. *Apply* $obdd_Y \cdot obdd_N$ *and the result is* $obdd_{Consistency}$.

A path with 1 as the terminal node in $obdd_{Consistency}$ indicates the inconsistency of the decision table.

In Section 5.2.1, the definition of consistency $DT_{Consistency}$ has been given. A decision table contains inconsistency, if there exists a pair of overlapping rules and

their action entries are different. Each path of an OBDD represents a rule. $obdd_Y$ is built based on the rules of the decision table. The action entries of these rules are `Y`. As a result, the paths of $obdd_Y$ with 1 as the terminal node represent the rules with the action entries `Y` in the decision table. This set of paths must have 0 as the terminal node in $obdd_N$. If one of these paths has 1 as the terminal node in $obdd_N$, then the decision table contains a contradiction. It is because those paths of $obdd_N$ with 1 as the terminal node represent rules of the decision table with the action entry `N`. This concept is expressed by the propositional formula $obdd_Y \wedge obdd_N$ and the corresponding equivalent boolean function is $obdd_Y \cdot obdd_N$.

In Section 4.3.3, the pair of inconsistent rules in the decision table (see Figure 4.11 on page 47) is found by using the OBLH tool (see Figure 4.15 on page 49). The steps of applying $AOBDD_{Consistency}$ to check the consistency of this decision table is illustrated in Figure 5.17. The OBDD in Figure 5.17(a) is built based on $R1$ and $R4$ because their action entry is `Y`. The results of step 2 are shown in Figure 5.17(b). This Figure is built based on $R2$, $R3$, $R5$ and $R6$. Figure 5.17(c) shows the result of the algorithm. There is a single path with 1 as the terminal node and the corresponding transformed decision rule is (`Y`, `N`, `N`, `N`, `N`). This is the same inconsistent rule shown in Figure 4.15 on page 49.

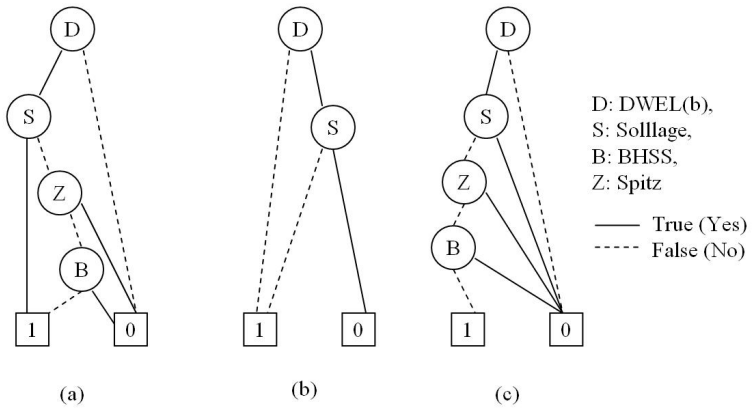


Figure 5.17: Inconsistency and OBDDs

5.6.2 Checking Completeness of Decision Tables

Given a consistent decision table DT, the completeness of DT is checked based on the following steps:

Algorithm 6 $AOBDD_{Completeness}$

1. *Build* $obdd_Y$ *based on the rules with* $a_{1j} =$ `Y`
2. *Build* $obdd_N$ *based on the rules with* $a_{1j} =$ `N`
3. *Apply* $obdd_Y + obdd_N$ *and the result is* $obdd_{Completeness}$.

This algorithm is called $AOBDD_{Completeness}$. A path with 0 as the terminal node in $obdd_{Completeness}$ indicates a missing rule of the decision table. If $obdd_{Completeness}$ has only 1 as the terminal node, the decision table is complete.

There are two possible representations of the paths in $obdd_Y$ that lead to the terminal node 0. The first possible representation are rules with action entries `N`. The second representation are missing rules. The rules in the first case are supposed to be found in $obdd_N$. If these rules cannot be found in $obdd_N$ as a path with 1 as the terminal node, then they are the missing rules and the decision table is incomplete. This concept is expressed by the propositional formula $obdd_Y \vee obdd_N$ and the corresponding equivalent boolean function is $obdd_Y + obdd_N$.

The missing rules of the decision table in Figure 4.10 on page 46 are found by using the OBLH tool. These missing rules are found in the OBLH tool by the implemented algorithm $AOBDD_{Completeness}$. Figure 5.18 illustrates the steps of $AOBDD_{Completeness}$ to find the missing rule of the decision table in Figure 4.10. The OBDD in Figure 5.18(a) is $obdd_Y$. $obdd_N$ is shown in Figure 5.18(b). The result of the algorithm $obdd_{Completeness}$ is illustrated in Figure 5.18(c). The path with 0 as the terminal node represents the missing rules. This path can be transformed into decision rules based on the algorithm described in Section 5.5.2. The missing rules are represented as a compact rule (`Y`, `N`, -, -, -) which is the same result as it has been shown in the OBLH tool (see Figure 4.13 on page 48).

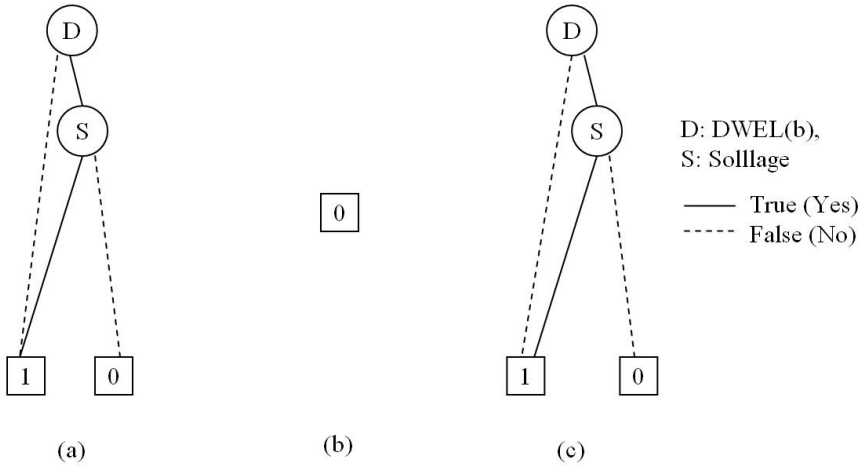


Figure 5.18: Checking completeness and OBDDs

5.6.3 Analysis and Verification of Decision Tables

Given decision tables DT_S and DT_C, the validity of the sequent $DT_S \models DT_C$ is checked based on the following steps:

Algorithm 7 $AOBDD_{Sequent}$

1. *From DT_S, build $obdd_S$ based on the rules with $a_{1j} =$* `Y`

2. From DT_C, build $obdd_C$ based on the rules with $a_{1m} =$ Y

3. Apply $\overline{obdd_S} + obdd_C$ and the result is $obdd_{Sequent}$.

$j \in \{1, ..., n\}$ and n: number of rules in the decision table DT_S
$m \in \{1, ..., q\}$ and q: number of rules in the decision table DT_C

Those paths with 0 as the terminal node in the resulting OBDD illustrate those evaluations of DT_S which cause the invalidity of the sequent.

As it has been shown in Section 5.1.3, situational analysis and checking the correctness of system requirements are defined as checking the validity of the sequent between the defined situations, checking conditions for verification and system requirements. These defined situations, checking conditions and system requirements are specified in decision tables and therefore, the algorithm $AOBDD_{Sequent}$ for checking validity of sequents in a form of decision tables is defined as above.

In propositional logic, checking the validity of a sequent (e.g. $S \models C$) means to check whether the implication between the premise and the conclusion holds ($S \rightarrow C \equiv \neg S \vee C$) (see Section 5.1.3). This can be achieved by the corresponding boolean function ($\overline{obdd_S} + obdd_C$) as it has been shown in step 3 of $AOBDD_{Sequent}$ when decision tables are used to represent the system requirements, a situation and checking conditions for verification.

The validity of the sequent is indicated by the paths (with 0 as the terminal node) of the resulting OBDD. As it has been mentioned before, it is very important to know the evaluations that cause the invalidity of the sequent. In case of verification of system requirements, these paths indicate the situations which are admissible under the system requirements, while are not admissible under the checking conditions (e.g. expected behavior of the system). For the analysis of system requirements, these paths indicate the inconsistent specifications between two system requirements. In the situational analysis, an existence of these paths means that the situation does not fulfill the system requirements.

In Section 4.3.2, the fulfillment of a situation w.r.t. $Requirement_{1.7}$ has been shown (see Figures 4.6 on page 44 and 4.7 on page 44). $AOBDD_{Sequent}$ is used by the OBLH tool to check this fulfillment. Figure 5.19 illustrates the corresponding steps of applying the algorithm. The OBDD $obdd_S$ on the left hand side of Figure 5.19(a) describes the specified situation. The OBDD on the right hand side is the negation of $obdd_S$. Based on the decision table of $Requirement_{1.7}$, $obdd_C$ is built and it is shown in Figure 5.19(b). In the final step, the boolean operation + is applied to $obdd_C$ and the negation of $obdd_S$. The result is shown in Figure 5.19(c). Since this resulting OBDD contains a path with 0 as the terminal node, the situation is invalid as it has been shown in the OBLH tool.

The fulfillment of another situation shown in Figure 3.2 has also been checked by using the OBLH tool (see Figures 4.8 on page 45 and 4.9 on page 45). The steps of the algorithm to check the fulfillment of this situation are shown in Figure 5.20. The OBDD $obdd_{S2}$ in the left hand side of Figure 5.20(a) describes the specified situation. The OBDD in the right hand side is the negation of $obdd_{S2}$. Based

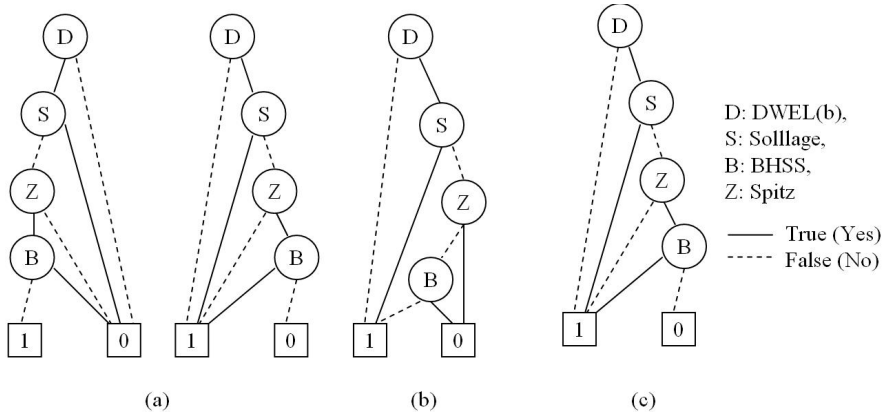


Figure 5.19: The situational analysis of a situation in Figure 3.1 and OBDDs

on the decision table of $Requirement_{1.7}$, $obdd_C$ is built and it is shown in Figure 5.19(b). In the final step, the boolean operation + is applied to $obdd_C$ and the negation of $obdd_{S2}$. The result is shown in Figure 5.20(b). Since this resulting OBDD contains no path with 0 as the terminal node, the situation is valid as it has been shown in the OBLH tool.

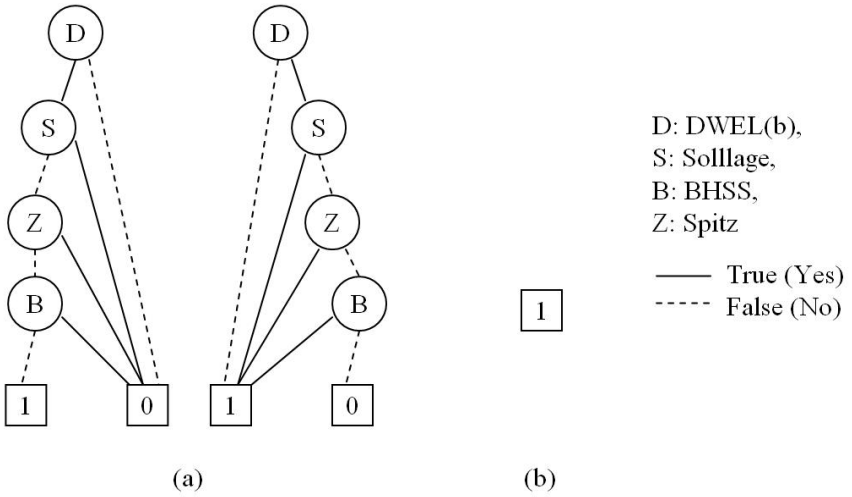


Figure 5.20: The situational analysis of a situation in Figure 3.2 and OBDDs

Chapter 6

Discussion

The motivation of this work has been transformed into a number of ideas to assist system development teams of different professions to specify, analyze and verify interlocking system requirements. These ideas and the corresponding implementations have been discussed informally and formally in previous chapters. This framework has been successfully applied in an industrial project to model a part of an interlocking system requirement specification and checked the correctness of this specification with the OBLH methods [HGS09]. A number of unclear system requirements was discovered and discussed in this project. Since no versatile framework can be developed without potential extensions and improvements, further development and investigation of OBLH are outlined based on this industrial experience. They are discussed in the following paragraphs.

Apart from the investigations on specification of dynamic conditions of the interlocking logic (see Section 4.2.2.2), an additional research idea is to provide users a chance to declare the dependencies of objects' attributes (physically unrealistic situations of railway operations) in decision tables, which can be called domain knowledge decision tables. Then, the OBLH tool can be used to discover the existence of these dependencies in system requirements (in the form of decision tables) and to check whether these dependencies have been stated correctly in the system requirements according to the domain knowledge.

Static conditions are defined by attributes of infrastructure elements in the form of propositional formulas or decision tables. Based on the properties of propositional formulas and decision tables, the combinations of these attributes define all the situations that the RIS needs to consider during the development of safe routes. If there exist dependencies among the attributes, some of these situations might be physically unrealistic situations (e.g. an element must not indicate the aspect Ks1 and Ks2 at the same time). These unrealistic situations must be declared as inadmissible in the interlocking logic. If the RIS receives such combinations during the development of a safe route, operational mistakes must appear among the infrastructure elements and the RIS must evaluate the development of this safe route as inadmissible. To maintain the correctness and completeness of the system requirements in the form of decision tables or propositional logic, these dependencies can be first declared by users as domain knowledge in deci-

sion tables (e.g. $DT_{Domainknowledge}$). These domain knowledge decision tables can be used to assist users in identifying complex dependencies in the static conditions (e.g. in form of decision table DT_{SR}) and check whether these dependencies are defined according to the declared domain knowledge by applying Method 4 ($DT_{SR} \models DT_{Domainknowledge}$).

In Figure 6.1, part of the domain knowledge of the signal aspects has been declared in $DT2$. According to the text of system requirement 5.2.2 of the specification [Tut06] (see Example 7 on page 18), the element satisfies the system requirement 5.2.2 if it shows the aspect Ks1. When this system requirement 5.2.2 is translated to a propositional formula and transformed into a decision table $DT1$, this situation is indicated in $R1$. Based on the checking result $DT3$ in Figure 6.1, this rule contains an operational situation which does not have the same result as it is declared in the domain knowledge decision table $DT2$. If a computer scientist reads the system requirement 5.2.2 and transforms it into a program of an RIS as it has been stated in this example, then an error will exist in the program. This mistake, of course, can be found out by the domain experts immediately when the decision table is read. It will then be specified correctly. However, when complex system requirements are defined in decision tables, decision tables can be complicated and contain many rules. With this concept, the OBLH tool can help to identify the dependencies of attributes w.r.t. the declared domain knowledge decision tables in complex decision tables and check whether they are defined according to the declared domain knowledge.

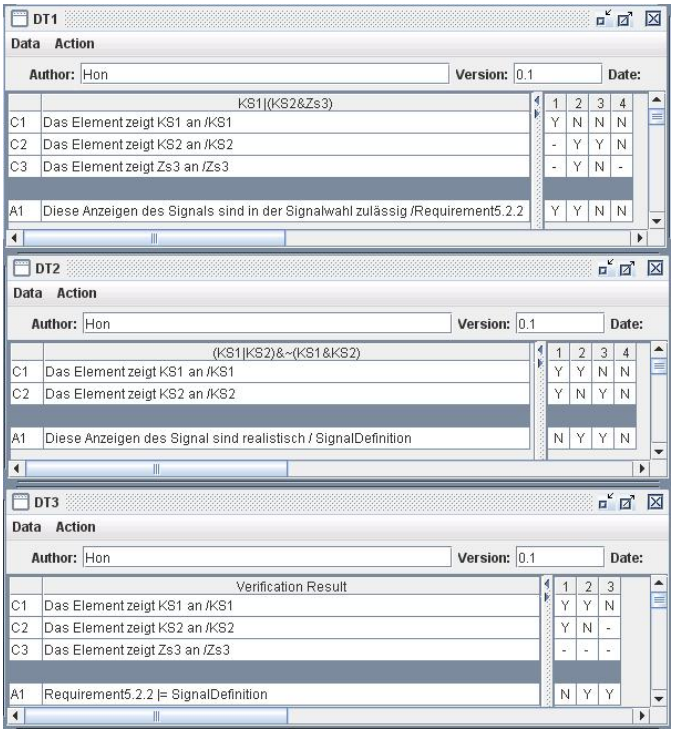


Figure 6.1: The comparison between the domain knowledge decision table and the system requirement 5.2.2, $Requirement_{5.2.2} \models SignalDefinition$

In the mentioned industrial project, it has been noticed that a system requirement specification (system requirement 3.3 of [Tut06] and see Table 6.1) in the specification consists of quantified statements. These statements can be modeled by quantifiers and predicates. Quantifiers are notations that are used in predicate logic [HR04, VOQ88]. There are two predicate quantifiers which are $\forall$ and $\exists$. They mean 'for all' and 'there exists at least one'. If a quantifier is used in combination with a predicate (an attribute which can be true or false), then it can be viewed in a similar way as an atomic formula in propositional logic. An example of an attribute or predicate is $FLSchutzbieten(w)$ which means the point located in the connected track section can provide flank protection (German: Die Weiche in dem anschließenden GFM (Gleisfreimeldeeinrichtung)-Abschnitt könnte Flankenschuzt bieten.). An example of combinations of a quantifier and an attribute is $\forall w \in Weichen\ FLSchutzbieten(w)$. This combination can be modeled as an atomic formula in propositional logic as $Alle_{FLSchutzbieten}$ which means all the points located in the connected track detection section can provide flank protection. (German: Alle Weichen in dem anschließenden GFM-Abschnitt können Flankenschutz bieten)

No.	System requirement	Static condition
3.3	*Umstellbedingung für nicht grenzzeichenfrei isolierte Weiche oder Kreuzung* *(a) Die am "umzustellenden Schenkel" der Weiche oder Kreuzung anschließende Weiche oder Kreuzung muss frei sein, wenn sie nicht die Flankenschutzlage einnimmt.* *Dies gilt für alle Weichen oder Kreuzungen, die mit der nicht grenzzeichenfrei isolierten Weiche oder Kreuzung eine gemeinsame Gleisfreimeldeeinrichtung haben.* *b) Schließen an den Schenkel der umzustellenden, nichtgrenzzeichenfrei isolierten Weiche mehrere Weichen mit einer gemeinsamen Gleisfrei-meldeeinrichtung an, die alle Flankenschutz bieten können, muss das Freisein nicht geprüft werden, wenn mindestens eine Weiche die Flankschutzlage einnimmt.*	$FreiGFM \vee$ $(\forall w \in Weiche\ FLSchutzbieten(w) \wedge \exists w \in Weiche\ FLSchutzeinnehmen(w))$

Table 6.1: The specification of system requirement 3.3

$FreiGFM$: The track section which is connected to a track section exposing the fouling point is free.
Der am nicht grenzzeichenfrei isolierten Schenkel anschließende GFM-Abschnitt ist frei.

$FLSchutzbieten(w)$:	The point located in the connected track section can provide flank protection. Die Weiche in dem anschließenden GFM-Abschnitt kann Flankenschutz bieten.
$FLSchutzeinnehmen(w)$:	The point located in the connected track section is in the flank protective position Die Weiche in dem anschließenden GFM-Abschnitt nimmt die Flankenschutzlage ein.

Based on the definition of requirement 3.3 alone (see Figure 6.2), a point with foul track circuit (German: nicht-grenzzeichenfrei isolierte Weiche) is allowed to be set to the requested position in the following two cases:

1. The track section which is connected to a track section exposing the fouling point is free.

2. All the points in this track section can provide flank protection and at least one of these points is in the flank protective position.

The requirement 3.3(a) is already specified in this predicate formula because this requirement is fulfilled if the track section which is connected to the point with foul circuit is free (point 1) or the point in this track section is in the flank protective position (point 2). The quantifiers in this formula refer directly to attributes, so that the corresponding static conditions can then be modeled by the specification language of OBLH, propositional logic. This is achieved by defining atomic propositions which express the meaning of $((\forall w \in WeicheFLSchutzbieten(w))$ and $(\exists w \in WeicheFLSchutzeinnehmen(w)))$, respectively. These atomic propositions are $Alle\_FLSchutzbieten$ and $Eins\_FLSchutzeinnehmen$, respectively. The predicate formula is transformed to a decision table in Figure 6.2. The system requirement 3.3 is specified as a propositional formula and the formula is transformed to a decision table in the OBLH tool and is shown in Figure 6.3.

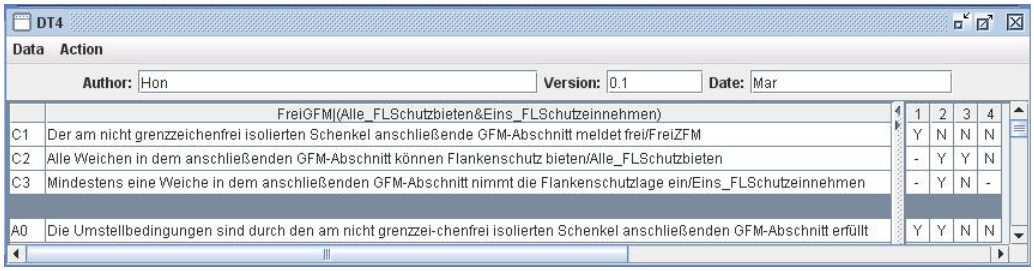


Figure 6.2: The meaning of the predicate formula of $Requirement_{3.3}$

As it has been shown in Figure 6.2, in the current version of the OBLH tool, a predicate formula with quantifiers can be transformed to a decision table. However, in this formula, the quantifier can only be used in combination with an attribute. A condition in a decision table is an atomic formula and the quantified attribute can be considered as an atomic formula. Therefore, predicate formulas in this form can be transformed to a decision table. If the quantifiers are used in combination

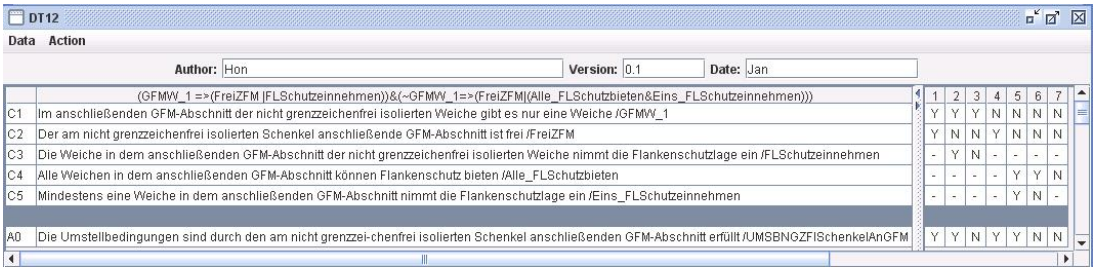


Figure 6.3: The meaning of the propositional formula of $Requirement_{3.3}$

with a complex formula, then the predicate formula cannot be transformed to a decision table in the current version of the OBLH tool.

There is another aspect that needs to be considered if a system requirement is specified by predicate logic. In predicate logic, the quantifiers $\forall$ and $\exists$ consist of a dependency relationship, for example, if the formula $(\forall w \in Weiche\ FLSchutzbieten(w))$ is fulfilled, it can also be deduced that $(\exists w \in Weiche\ FLSchutzbieten(w))$ is evaluated to be true[1]. If a system requirement is specified with these two formulas at the same time, then the corresponding dependency must also be correctly expressed. It has been mentioned above, if dependencies exist between attributes of objects in a decision table, then system requirements must be checked against the declared dependencies. The same concept can then be applied to the dependencies between quantified attributes. Users specify the dependencies in a decision table $DT_{Dependency}$ (see Figure 6.4). The system requirements in form of a decision table DT_{SR} are then verified against $DT_{Dependency}$ (e.g. $DT_{SR} \models DT_{Dependency}$). This can help to ensure the correctness of the specification w.r.t. the dependency of quantified attributes.

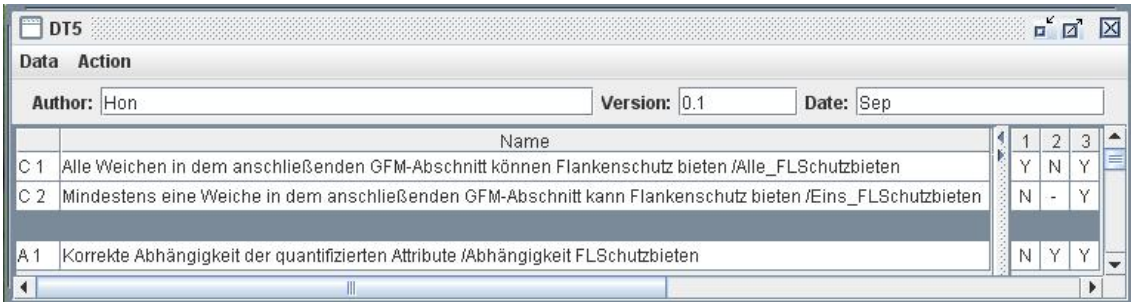


Figure 6.4: An example of a dependency between quantified attributes

It should be noticed that system requirements in a form, like system requirement 3.3, rarely appear in interlocking system requirements specifications. Therefore, the necessity of extending OBLH to support specification of system requirements in predicate logic shall be estimated before additional considerations in this aspect are made.

In the current version of the OBLH tool, the function for defining the meaning of each object's attribute is not supported. This means, when a propositional

[1] $|Weichen| > 0$

formula is transformed to a decision table, users need to type in the meaning of each attribute manually. Therefore, the OBLH tool can be further improved by providing such a function. Users define the meaning of each attribute once in the tool. Rather than a proposition (e.g. $BHSS$), it will be indicated as the meaning (e.g. the object is equipped with a movable frog or in German: Das Element ist mit beweglicher Herzstückspitze ausgerüstet /BHSS) in the corresponding condition stub in the decision table depending on the choice of users.

The last point that needs to be discussed is the novelty of the ideas of this formal framework. Specifying system requirements in decision tables or in tabular forms and checking completeness and consistency of decision tables has been addressed in a number of research projects [VD94b, HZ95, HL96, Kos06]. However, they are different from OBLH in a number of aspects. Among these projects, well-defined OBLH concepts like transformation of propositional logic, situational analysis and checking correctness of system requirements with decision tables and the corresponding implementation are not found. Secondly, decision tables or tabular forms and the algorithms for implementation are not formally defined in some of these projects. Therefore, the correctness of the implementations are difficult to be analyzed, for example, in [Kos06], checking completeness and consistency of decision tables has been discussed and implemented. However, decision tables and algorithms of the implementation are not formally defined in this work. Without a formal definition, well-defined mathematical packages, like the OBDD package cannot be used. The correctness of the checking results cannot be guaranteed. In some of these projects, workbenches are developed for specifying the system requirements. However, they are not as user-friendly as the OBLH tool, for example, in Prologa (Procedural Logic Analyzer) [VD94b, Van91, Van08], instead of specifying system requirements directly in a decision table, decision rules need to be typed in the tool. They are then transformed and represented in decision tables.

Chapter 7

Conclusion

In this work, a part of concepts of German interlocking systems was first introduced and the properties of their specifications were discussed. It was stated in Section 2.3 that these specifications are written in German. Requirements are outlined by complex railway terms and concepts. Furthermore, similar and even identical system requirements are formalized by different statements in different positions within the specifications. As a result, some system requirements are difficult to be interpreted. There is also a high chance to misinterpret the same system requirement by different professions in the system development teams. Since the system requirements are written in natural language, their correctness is difficult to be checked. Inaccurate and wrong specifications affect the efficiency of system developments and might cause errors in these systems. The later the discovery of such mistakes is, the higher the costs of system developments are. An error in a safety critical system, like RIS, might even cost human life.

In this work, applying formal methods in the early stage of an interlocking system development is the proposed solution to handle these problems. The reasons were discussed in detail in Section 3.1. The meaning of a formally specified requirement becomes clear. The existence of errors in formal specifications can also be checked by applying the corresponding mathematical concepts in the early phases of system developments, such that no errors will be transferred to the next phases of system developments. However, applications of formal methods have not been widespread in the railway domain. As it was discussed in Section 3.2, the main reason is the lack of participation and trust of railway engineers in applying formal methods. Therefore, the developed formal framework in this work is engineering-oriented, such that different professions can also profit from the usage of formal methods.

In chapter 4, the main concepts of OBLH and the OBLH tool were introduced. With these engineering-oriented concepts and this implemented tool, different professions of system development teams can specify, analyze and verify interlocking system requirements formally. These system requirements can be classified based on their nature and the defined categories in OBLH. Specifications become then well-structured. Task assignment to each system development team and verification of specifications becomes easy and efficient. The suitable formal specification

language, propositional logic, is used to model an important part of interlocking system requirements – the checking conditions that an RIS needs to consider during the development of a safe route. These conditions are called static conditions in OBLH. With the usage of propositional logic, the meaning of each static condition becomes explicit and precise.

In this framework, users without a strong mathematical background, can also understand the meaning of each formal requirement by using the function of transformation. A propositional formula is transformed to a truth table and a decision table. Its meaning is represented in these tables. The formal and clear system requirements can then be understood easily and analyzed by different experts of system development teams. When a complex requirement is expressed in the form of a decision table, users can achieve the assessment of a situation w.r.t. interlocking system requirements with the function called situational analysis in OBLH (see Section 4.3.2). The analysis of a situation can be carried out quickly and automatically in the OBLH tool.

Among the usage of a representation form of propositional formulas, decision tables can also be used to specify static conditions in OBLH. These decision tables must be complete and their rules must be consistent among each other. In order to facilitate specification of well-formed decision tables, concepts of checking consistency and completeness w.r.t. the specified conditions of decision tables are also developed and implemented in the OBLH tool (see Section 4.3.3).

One of the attractive benefits of applying formal methods is finding wrongly specified requirements. Based on the discussed properties of interlocking system requirements in Section 2.3, mathematics-based methods for specifying, analyzing and verifying these requirements were defined in OBLH (see Section 4.3.4). In the OBLH tool, the system requirements, expected behaviors and safety requirements for verification and analysis are specified in decision tables. The corresponding mathematical operations of each method are then carried out automatically in the OBLH tool. As a result, railway engineers can also easily analyze and verify the system requirements in this framework.

In a formal framework, every component should be explicitly stated. The concepts and methods of OBLH were therefore mathematically defined by propositional logic in Chapter 5. In this chapter, propositional logic, decision tables and the OBDDs technique were formally introduced. Their relationships and applications in OBLH were also discussed. The OBLH tool was implemented based on these formal concepts. With these concepts, the implementation can be easily achieved and understood. The correctness of their results can also be guaranteed.

With the current defined concepts of OBLH and the implemented OBLH tool, different professions can already state and examine the interlocking system requirements formally, although a number of aspects of this formal framework can be further improved (see Chapter 6). This framework was successfully applied to specify and check the correctness of a part of an interlocking requirement specification in an industrial project [HGS09]. In this project, these natural language written system requirements are specified in propositional logic and transformed to decision tables in the OBLH tool. It was shown that system requirements can

be clearly stated and well-understood in both propositional formulas and decision tables. Railway engineers and computer scientists can analyze the formal requirements without any difficulty. No misinterpretation of specifications can be caused. Furthermore, unclear specifications were discovered and discussed. The correctness of these system requirements is then increased after the application of OBLH. OBLH can also be used in a different way. Instead of writing interlocking system requirements in a natural language, requirements specifiers, like railway engineers, can specify interlocking system requirements in decision tables directly and these decision tables can be analyzed and verified in the OBLH tool. In this way, specifiers can also benefit from the mentioned and demonstrated advantages of the industrial project.

To summarize, the formal framework OBLH successfully brings the advantages of utilization of formal methods to each member of interlocking system development teams. More importantly, railway engineers are supported to play an important role in using this formal framework. By applying this framework in the early stage of the system development process, the specifications become precise and consistent. As it was stated in different chapters of this work, the whole development process becomes then efficient and cost-effective and the number of errors in the system becomes less.

Appendix A

List of Symbols

Logic

Symbol	Symbol Meaning
$\because$	because
$\neg\phi$	not ϕ (logical negation)
$\phi \wedge \psi$	ϕ and ψ (logical and)
$\phi \vee \psi$	ϕ or ψ (logical or)
$\phi \rightarrow \psi$	if ϕ is true, then ψ must be true (logical implication)
$\phi \equiv \psi$	ϕ and ψ are logically equivalent (logical equivalence)
$\phi \models \psi$	ψ is the logical consequence of ϕ semantically
$\phi \vdash \psi$	ψ is the logical consequence of ϕ syntactically
$\forall$	for all
$\exists$	there exists

Set theory

Symbol	Symbol Meaning
$\{x_1, x_2, ..., x_n\}$	a set consisting of the elements $x_1, x_2, ..., x_n$, n is the number of elements of the set
$\|X\|$	the number of elements in the set X
$x \in X$	x is an element of the set X
$X \subseteq Y$	X is a subset of Y
$X \backslash Y$	all elements from X that are not in Y
$x \neq y$	the elements x and y are not the same
$\{\texttt{Y}, \texttt{N}\}^*$	all possible sequences composed of the elements `Y` and `N`

Backus Naur Form (BNF)

Symbol	Symbol Meaning
$Exp := Rule_1$	the well-formed expression Exp can be composed by applying expression rule $Rule_1$
$Rule_1 \mid Rule_2$	a well-formed expression of a language can be formed by applying expression rules $Rule_1$ or $Rule_2$
$Exp_1 \Rightarrow Exp_2$	the expression Exp_1 is transformed into the expression Exp_2 by applying one of the expression rules given in BNF

Logical symbols from the OBLH tool[1]

Symbol	Symbol Meaning
$\sim \phi$	not ϕ (logical negation)
ϕ & ψ	ϕ and ψ (logical and)
$\phi \mid \psi$	ϕ or ψ (logical or)
$\phi => \psi$	if ϕ is true, then ψ must be true (logical implication)

[1]The propositional connectives $\wedge$, $\vee$, $\neg$ and $\rightarrow$ are represented as &, |, $\sim$ and => in the OBLH tool, respectively, because these propositioanl connectives are not found in the computer keyboard.

Appendix B

List of Abbreviations

Abbrevation	**Abbrevation Meaning**
BDDs	Binary Decision Diagrams
CENELEC	Comité Européen de Normalisation Electrotechnique
CSP	Communicating Sequential Processes
CTL	Computation Tree Logic
DNF	Disjunctive Normal Form
DWeg	Durchrutschweg
`F`	False
FL	Flankenschutz
FüMBli	Fahrstraßenfestlegeüberwachungsmelder Blinklicht
FüMR	Fahrstraßenfestlegeüberwachungsmelder Ruhelicht
FüM	Fahrstraßenfestlegeüberwachungsmelder
FW	Fahrweg
GFM	Gleisfreimeldeeinrichtung
GUI	Graphical User Interface
iff	If and only if
Ks	Kombinationssignal
LH-ESTW-R	Lastenheft fü das Elektronische Stellwerk Regional
LTL	Linear Time Logic
`N`	No
OBDDs	Ordered Binary Decision Diagrams
OBLH	Object-Based Lastenheft
RBDDs	Reduced Binary Decision Diagrams
RIS	Railway Interlocking System
RSL	Rigorous Approach to Industrial Software Engineering Specification Language
Prologa	Procedural Logic Analyzer
SAT	Satisfiability Solver
SMV	Statistical Model Validation
`T`	true
UMBP	Prüfung der Umstellbedingungen
UML	Unified Modeling Language

Abbrevation	**Abbrevation Meaning**
VDM	Vienna Development Method
wff	well formed formulas
w.r.t.	with respect to
Y	Yes
Z	Z notation
ZPZ	Zulassungsprüfung Zugstraße
Zs	Zusatzsignal

Appendix C

Railway Terms

This appendix lists German railways terms and the corresponding possible English translation [Wor08, Pac08].

German	**English**
Anzeige der Fahrdienstleiterarbeitsoberfläche	Graphical User Interface(GUI) of the RIS
Ausfahrstraße	Safe exit route
Bahnhof	Home signals limits
Beansprucht	locked or reserved
Befahrbarkeitssperre	A track section blocked for train movements
Bewegliche Herzstückspitze	Movable frog
Freie Strecke	Open line
Deutsche Bahn	German Rail
Durchfahrstraße	Non-stop route
Durchrutschweg/DWeg	Overlap
D-Weg-Ziel	Overlap destination
Einfahrstraße	safe entrance route
Fahrdienstleiter	train director
Fahrstraße	Safe route
Fahrstraßenlogik	Interlocking logic
Fahrweg	Route
Fahrwegelement	Route element
Fiktiver Zielpunkt	Virtual route destination's point
Flankenschutz	Flank protection
Flankenschutzelement	Flank protection element
Gegenfahrausschluss	Locking of conflicting routes from the opposite direction
GFM-Abschnitt	Track section
Gleis	Track
Grundsatz	Basic condition
Infrastrukturelement	Infrastructure element
Kreuzung	Crossing

German	**English**
Lage	The position of the point
Rangierstraße	Shunting route
Reservieren	Reserved
Nicht grenzzeichenfrei isolierte	Point with fouling track circuit
Weiche	Point
Schnittstelle Bahnhof/Strecke(Bf/Str)	Interface Bf/Str
Sicherheitsregeln	Safety requirements
Spitz befahren	Facing point movement
Start	Start
Stammgleis	Straight track
Stellwerk	Railway interlocking system
Stellwerkslogik	Interlocking system logic
Streckenblockabschnitt	Block section
Streckenblock	Automatic block system
Stumpf befahren	Trailing point movement
Überlappende D-Wege	Overlapping overlaps
Umstellsperre	Manually locking points
Umstellbedingung	Condition for moving points
Verschließen	Lock
ZPZ-Bedingungen	ZPZ conditions
ZPZ-Bedingungen je Fahrwegelement im Durchrutschweg	ZPZ conditions for an infrastructure object located within the overlap
ZPZ-Bedingungen je Fahrwegelement im Fahrweg	ZPZ conditions for an infrastructure object located within the route
ZPZ-Bedingungen für die gesamte Zugfahrstraße	ZPZ conditions for a safe route
ZPZ positiv	ZPZ positive
Zugfahrstraße/ Zugstraße	safe route
Ziel	Route destination
Zweiggleis	Diverging track

Appendix D

Summary of Objects' Attributes

$Alle\_FLSchutzbieten$:	All points located in the track section can provide flank protection. Alle Weichen in dem anschließenden GFM-Abschnitt können Flankenschutz bieten.
$BHSS$:	The object is equipped with a movable frog. Das Element ist mit beweglicher Herzstückspitze ausgerüstet.
Bsp:	The object is blocked for train movements. Eine Befahrbarkeitssperre ist gesetzt.
$EL(Durchfahrt)$:	The locking or reservation belongs to a part of the requested non-stop safe route. Nutzung gehört zu einem Teil der einzustellenden Durchfahrt.
$DWEL(b)$	The object is locked or reserved as an overlap element of other routes. Das Element ist als DWeg-Element (DW$\geq$0) für eine oder mehrere Zugstraßen beansprucht.
$DWEL(v)$:	The object is locked as an overlap element of other routes. Das Element ist als DWeg-Element (DW$\geq$0) für eine oder mehrere Zugstraßen verschlossen.
$Eins\_FLSchutzeinnehmen$:	There exists at least one point located in the connected track section is in the flank protective position. Mindestens eine Weiche in dem anschließenden GFM-Abschnitt nimmt die Flankenschutzlage ein.

$EL(v)$:	The object is locked as a route, an overlap or a flank protection element of other routes. Als Fahrwegelement, DWeg-Element oder Flankenschutzelement für eine andere Zugstraße verschlossen.
$FLEL(b)$	The object is locked or reserved as a flank protection element of other routes. Das Element ist als Flankenschutzelement für eine oder mehrere Fahrstraßen beansprucht.
$FLSchutzbieten(w)$:	The point located in the connected track section can provide flank protection. Das Element in dem anschließenden GFM-Abschnitt kann Flankenschutz bieten.
$FLSchutzeinnehmen(w)$	The point located in the connected track section is in the flank protective position. Das Element in dem anschließenden GFM-Abschnitt nimmt die Flankenschutzlage ein.
$Frei$:	The object of the requested safe route is free. Das für die Fahrstraße benötigte Element ist frei.
$FreiGFM$:	The track section which is connected to a track section exposing the fouling point is free. Der am nicht grenzzeichenfrei isolierten Schenkel anschließende GFM-Abschnitt ist frei.
$FLSchutzbieten(w)$:	The point located in the connected track section can provide flank protection. Die Weiche in dem anschließenden GFM-Abschnitt kann Flankenschutz bieten.
$FLSchutzeinnehmen(w)$:	The point located in the connected track section is in the flank protective position. Die Weiche in dem anschließenden GFM-Abschnitt nimmt die Flankenschutzlage ein.
$FWEL(b)$:	The object is locked or reserved as a route element of other routes. Das Element ist als Fahrwegelement für eine andere Fahrstraße beansprucht.
$KS1$:	The object indicates Ks1. Das Element zeigt Ks1 an.
$KS2$:	The object indicates Ks2. Das Element zeigt Ks2 an.
Usp:	The object is a manually locking point. Das Element ist gegen Umstellen gesperrt.

$Solllage$:	The object is in the proper position for the requested safe route. Das Element liegt in der Solllage für die einzustellende Zugstraße.
$Spitz$:	The requested overlap of the safe route is a facing point movement. Der einzustellende DWeg läuft über eine spitz zur Durchrutschrichtung liegende Weiche.
$Zs3$:	The object indicates Zs3. Das Element zeigt Zs3 an.
ZPZ_{Dweg}:	The evaluation ZPZ conditions of the requested overlap is positive. ZPZ des einzustellenden DWeg ist positiv.
$ZPZ_{DwegElement}$:	The evaluation ZPZ conditions of the requested overlap is positive (based on the ZPZ conditions in the view of an object). ZPZ des einzustellenden DWeg ist positiv (nach den ZPZ-Bedingungen auf der Ebene "Fahrwegelement im Durchrutschweg").
$ZPZ_{DwegGesamtzugstraße}$:	The evaluation ZPZ conditions of the requested overlap is positive (based on the ZPZ conditions in the view of a safe route). ZPZ des einzustellenden DWeg ist positiv (nach den ZPZ-Bedingungen für die Ebene "gesamte Zugfahrstraße").

Appendix E

OBLH User Guide

This appendix provides a guide for using the functions of the OBLH tool.

E.1 Starting the OBLH Tool

1. Double click the icon *OBLH*.
2. A workspace called *Object-Based Lastenheft* pops up in the desktop (see Figure E.1).

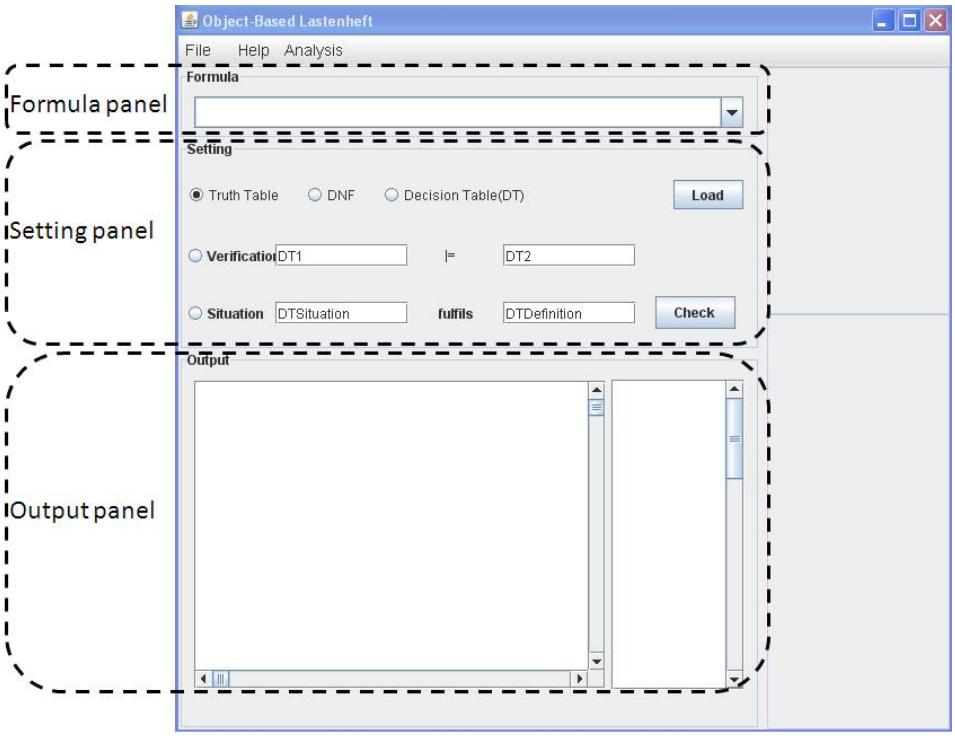


Figure E.1: The OBLH tool and start icon

E.2 Transformation of Formulas

In the OBLH tool, a formula can be transformed into different forms for analysis. Users can transform this formula into a truth table, a Disjunctive Normal Form (DNF) and a decision table. The theoretic background of the transformation was discussed in Section5.5.

E.2.1 Logical Connectives in the OBLH Tool

The OBLH tool provides a set of logical connectives to users to specify propositional formulas[1]. The logical connectives and the corresponding symbols are written as follows:

- Logical negation $\sim$
- Logical and &
- Logical or |
- Logical implication =>

If the input formula is not a well-formed formula, then a message box *Warning* pops up (see Figure E.2). It indicates the errors.

E.2.2 Formulas to Truth Tables

1. Type in the formula into the combobox in the panel *Formula*.
2. Click the button *Truth table* in the panel *Setting*.
3. Click the button *Load*.
4. The result is shown in the text area in the panel *Output* (see Figure E.3).

E.2.3 Formulas to DNF

1. Type in the formula into the combobox in the panel *Formula*.
2. Click the button *DNF* in the panel *Setting*.
3. Click the button *Load*.
4. The result is shown in the text area in the panel *Output* (see Figure E.4).

[1]The propositional connectives $\wedge$, $\vee$, $\neg$ and $\rightarrow$ are represented as &, |, $\sim$ and => in the OBLH tool, respectively, because these propositioanl connectives are not found in the computer keyboard.

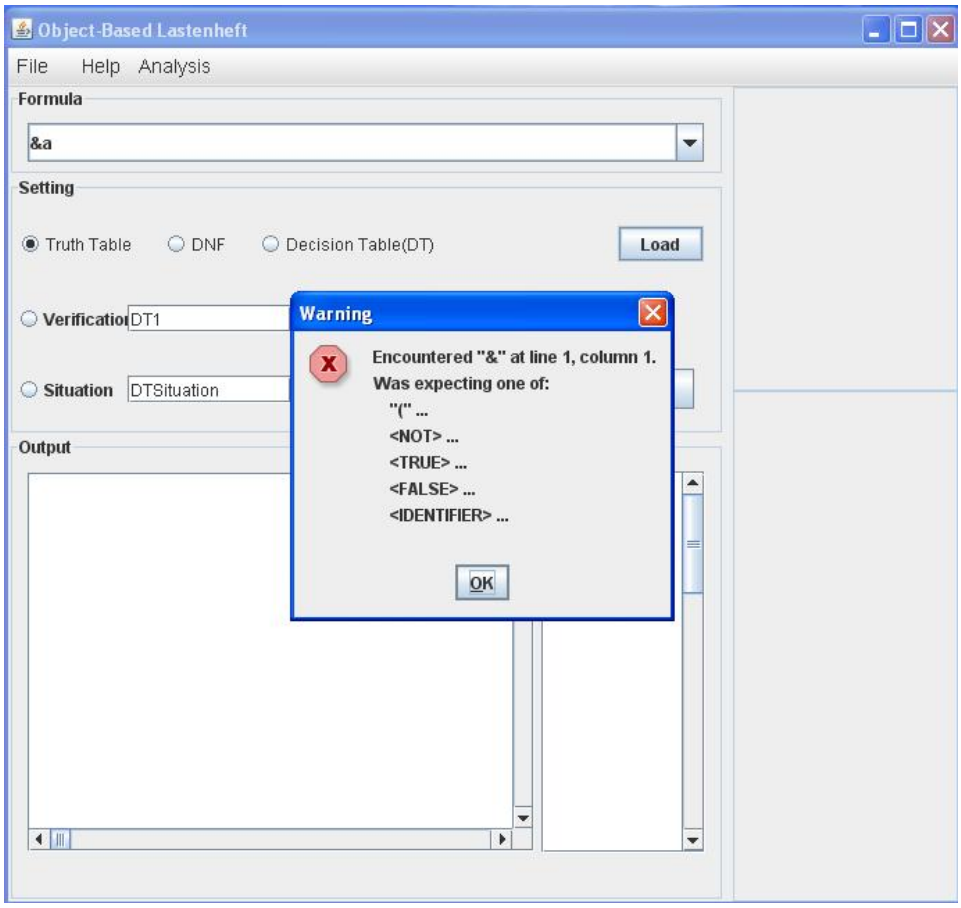


Figure E.2: An error message of non-wff

E.2.4 Formulas to Decision Tables

1. Type in the formula into the combobox in the panel *Formula*.
2. Click the button *Decision Table (DT)* in the panel *Setting*.
3. Click the button *Load*.
4. A window *DT* pops up. The name of the decision table is given based on the number of generated decision tables. For example, the first generated decision table is called *DT1*, while the second one is *DT2* (see Figure E.5).

E.3 Specification in Decision Tables

The window of the decision table (see Figure E.6) can be adjusted into different sizes. This can be achieved by stretching the corner of the window or using the icons *minimize* or *maximize*. They are located at the right-hand corner of the window. Furthermore, the size of the panel *Condition and action* can be changed by shifting the split bar.

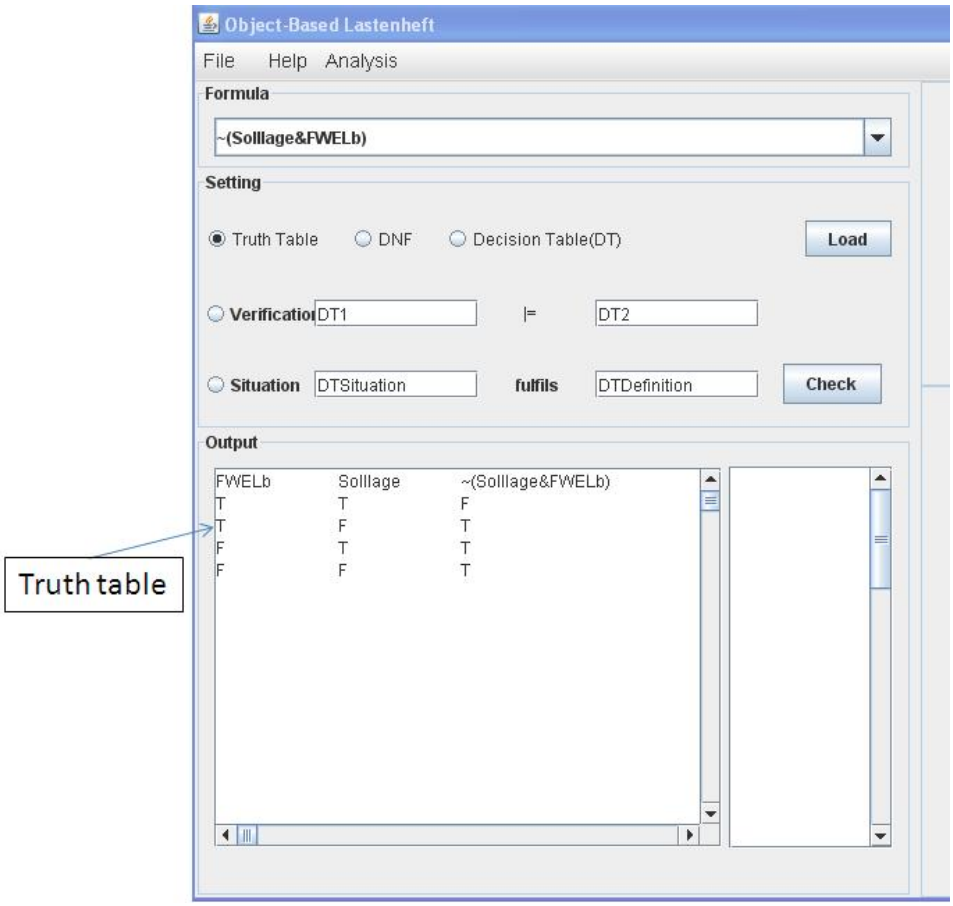


Figure E.3: The generated truth table

E.3.1 Create Decision Tables

1. Click the items *File>New>Decision Table* in the menu bar.
2. A window *DT* pops up.

E.3.2 Open Decision Tables

1. Click the items *File>Open>Decision Table (CVS)* in the menu bar.
2. Choose the file in the file window and click the button *Open*.
3. A window *DT* pops up.

E.3.3 Manipulation in Decision Tables

The functions that are provided to manipulate decision tables are written as follows:

- Add actions or conditions: Click the items *Action>Add Action* or *Action>Add Condition* in the menu bar.

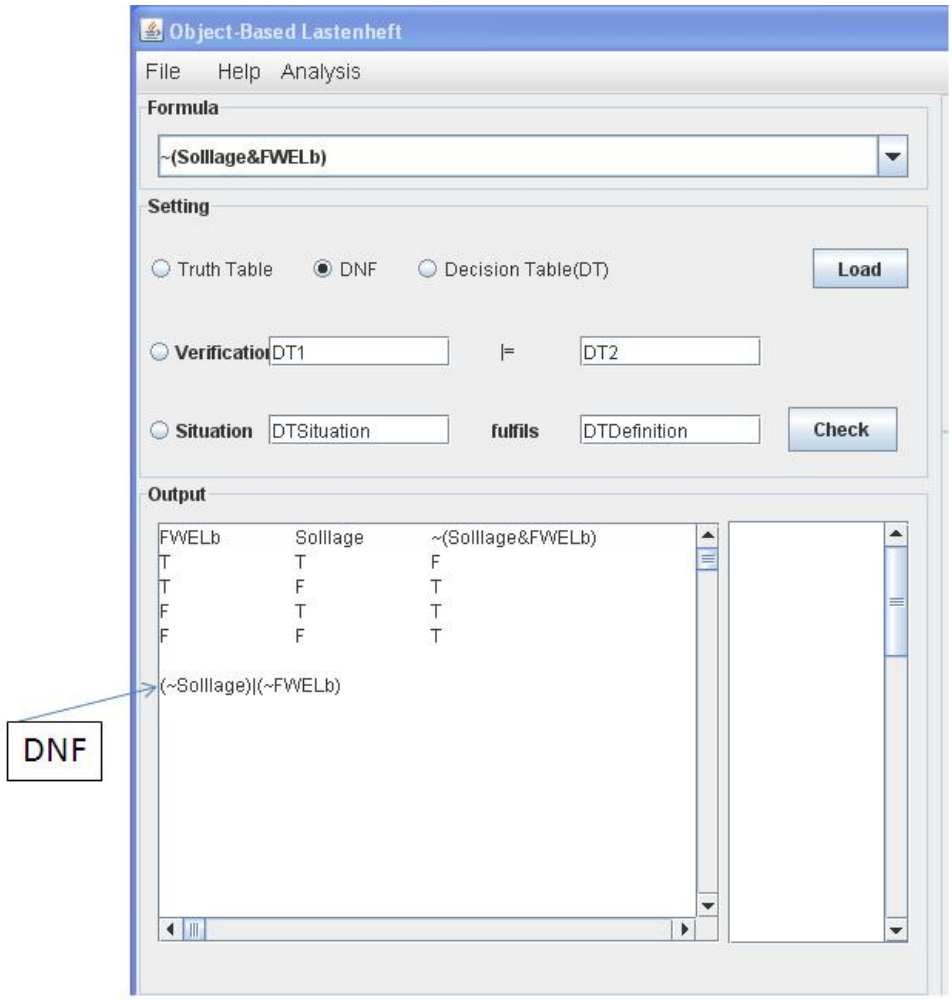


Figure E.4: The generated DNF

- Create rules: Click the items *Action>Create Rule* in the menu bar.
- Delete actions or conditions: Click the Action or Condition in the decision table that is needed to be deleted. Click the items *Action>Add Delete Action/Condition* in the menu bar.
- Delete rules: Click the items *Action>Delete Rule* in the menu bar. Type in the assigned number of the rule that is needed to be deleted in the pop-up message box. For example, deleting the rule *R5*, 5 is typed into the message box.
- Save decision tables: Click the items *Data>Store*. Type in the file name in the file window and click Save.
- Print decision tables: Click the items *Data>Print*.
- Close decision tables: Click the items *Data>Close* and save the decision table if it is needed. Or click the icon *Cross* that is located at the right-hand corner of the decision table window.

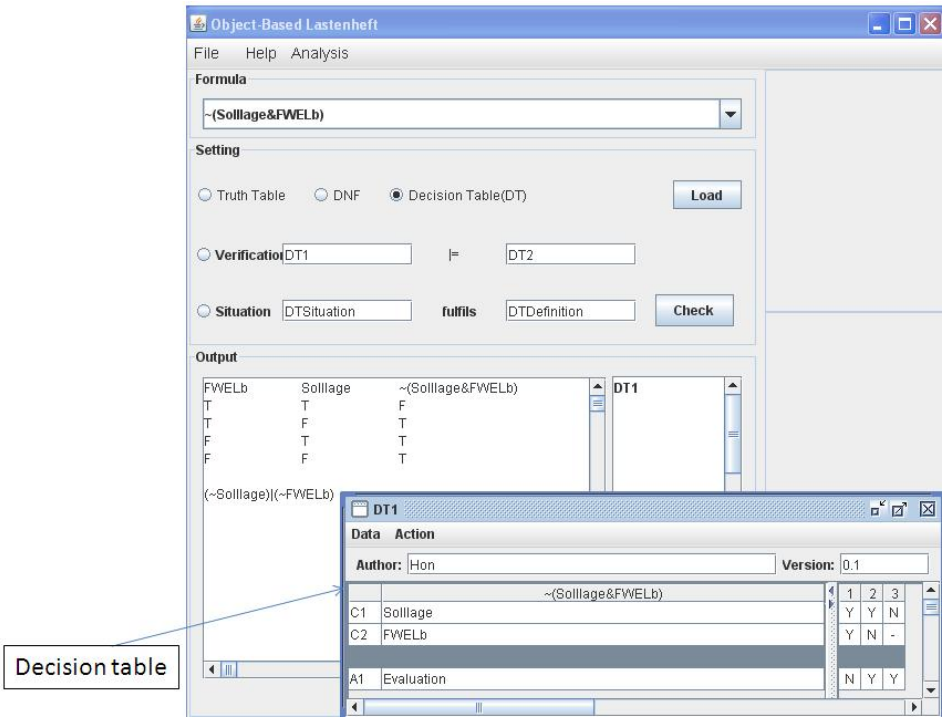


Figure E.5: The generated decision table

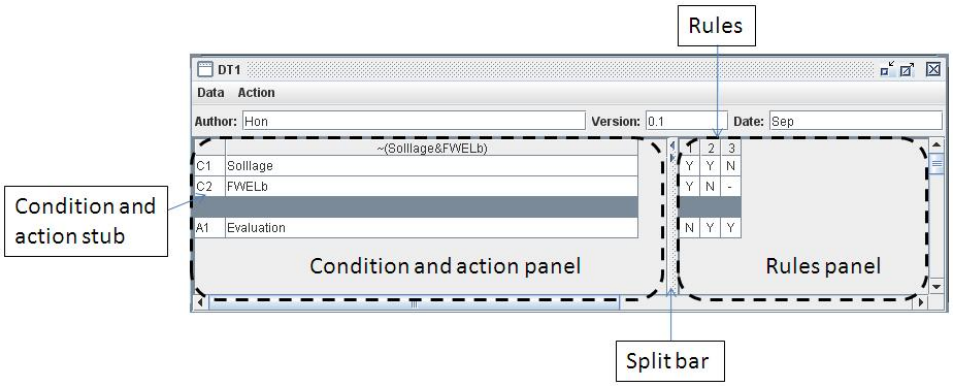


Figure E.6: The window of a decision table

E.4 Analysis of Specifications

The OBLH tool provides different ways to analyze a specification. First, the consistency of the user's input decision tables can be checked. Furthermore, missing rules of a decision table can be found out by checking its completeness. Finally, when a definition of the domain specific concept has been specified as a formula or a decision table, a situation can be evaluated based on the definition.

E.4.1 Checking Consistency of Decision Tables

1. Create and specify the decision table.

2. Click the name of the decision table in the list of decision tables that is located in the right-hand side of the panel *Output*.

3. Click the items *Analysis>CheckDTConsistency* in the menu bar. An error message window pops-up if there exists an empty field in the decision table.

4. A message box pops-up to indicate the consistency of rules of the decision table.

5. If an inconsistency is found in the decision table, an error message box pops-up and the contradicted rule(s) will be automatically generated in the decision table. Users need to specify the value of its(their) action(s) (in figure E.7).

E.4.2 Checking Completeness of Decision Tables

1. Create and specify the decision table (see Figure E.8).

2. Click the name of the decision table in the list of decision tables that is located in the right-hand side of the panel *Output*.

3. Click the items *Analysis>CheckDTCompleteness* in the menu bar. An error message window pops-up if there exists an empty field or inconsistency in the decision table.

4. A message box pops-up to indicate the completeness of the decision table.

5. If the decision table is not complete, an error message box pops-up and the missing rule(s) will be automatically generated in the decision table. Users need to specify the value of its(their) action (see Figure E.9).

E.4.3 Situational Analysis

A situation fulfills a system requirement or the system requirements if its combination of conditions is evaluated to positive based on the definition.

1. The decision table of the definition must be first generated (e.g.*DTDefinition*). See section E.2.4 and E.3.1[2].

2. To specify the situation, click the name of the decision table *DTDefinition* in the list of decision tables.

3. Click the items *Analysis>SpecifySituation*. A window pops-up. This decision table (e.g.*DTSituation*) consists of all the conditions of the definition that are required to describe the situation (see Figure E.10).

4. Click the button *Situation* in the panel *Setting*.

[2]Whenever the data in the decision table has been amended, click the grey bar properly before further analysis is performed.

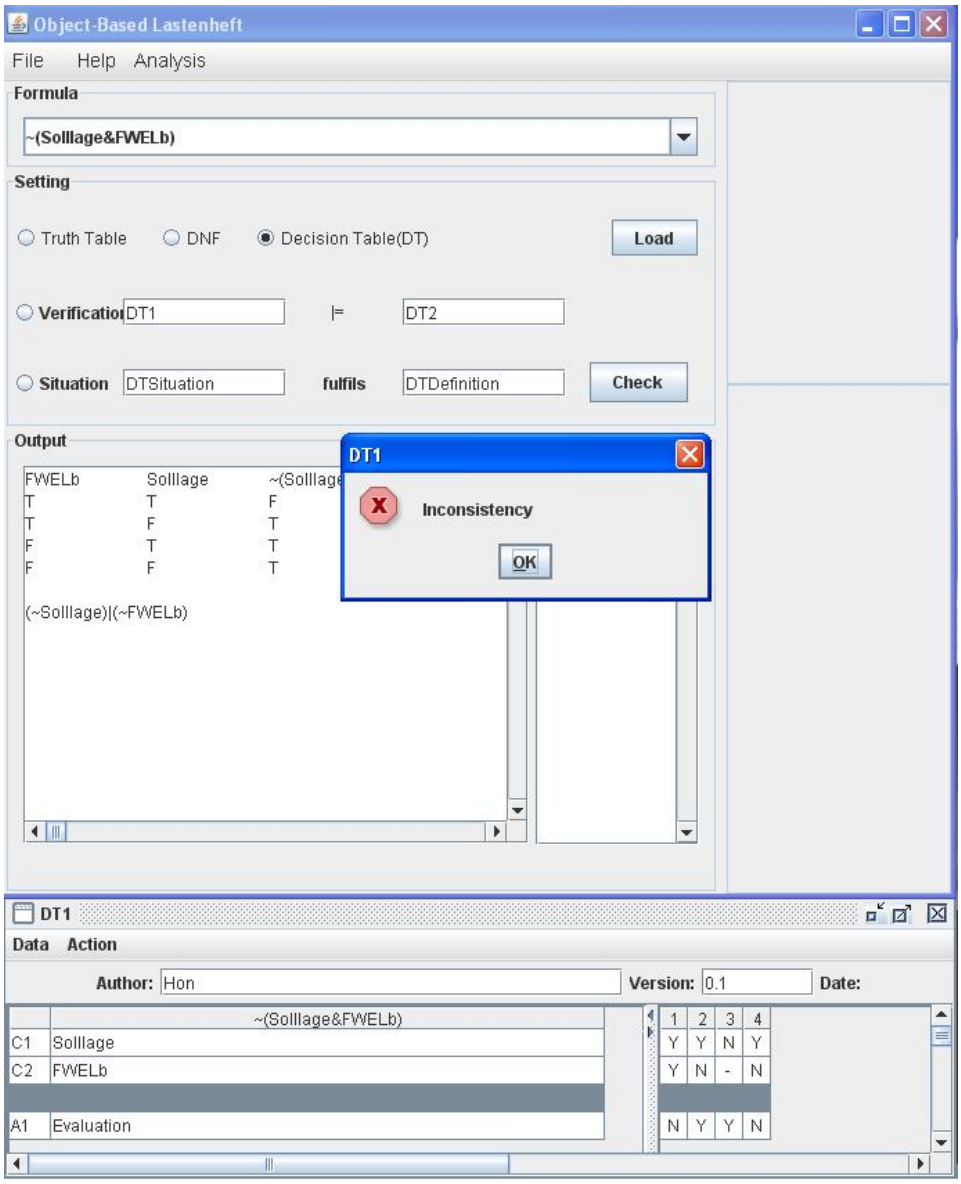


Figure E.7: The result of checking inconsistency

5. Type in the name of the decision table *DTSituation* into the left input box of the panel *Situation* (e.g.before the word *fulfills*)

6. Type the name of the decision table *DTDefinition* in the right input box of the panel *Situation* (e.g.after the word *fulfills*).

7. Click the button *Check*.

8. A message box pops-up to indicate the validity of the situation.

9. If the situation is invalid, a decision table is generated. In this decision table, rules with action `N` indicate rules of *DTDefinition* that the situation does not satisfy (see Figure E.11).

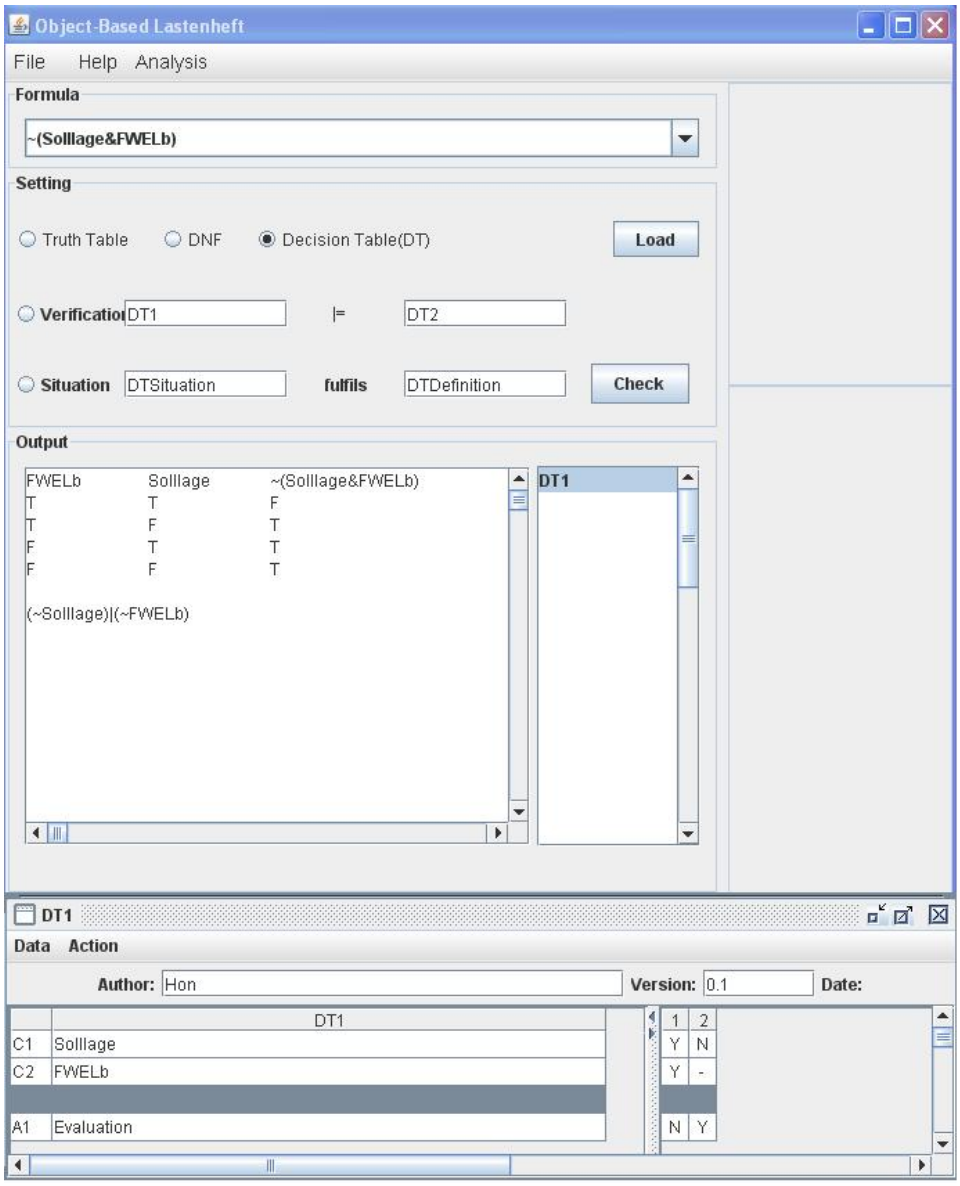


Figure E.8: Checking completeness

E.5 Logical Consequence of Formulas

This logical consequence of two formulas $\phi \models \psi$ can be checked in the OBLH Tool as follows:

1. The decision tables of these two formulas ϕ and ψ must be first generated [3].
2. Click the button *Verification*.
3. Type in the name of the decision table of ϕ (e.g.DTϕ) in the left input box of the panel *Verification* (see Figure E.12).

[3]Whenever the data in the decision table has been amended, click the grey bar properly before further analysis is performed.

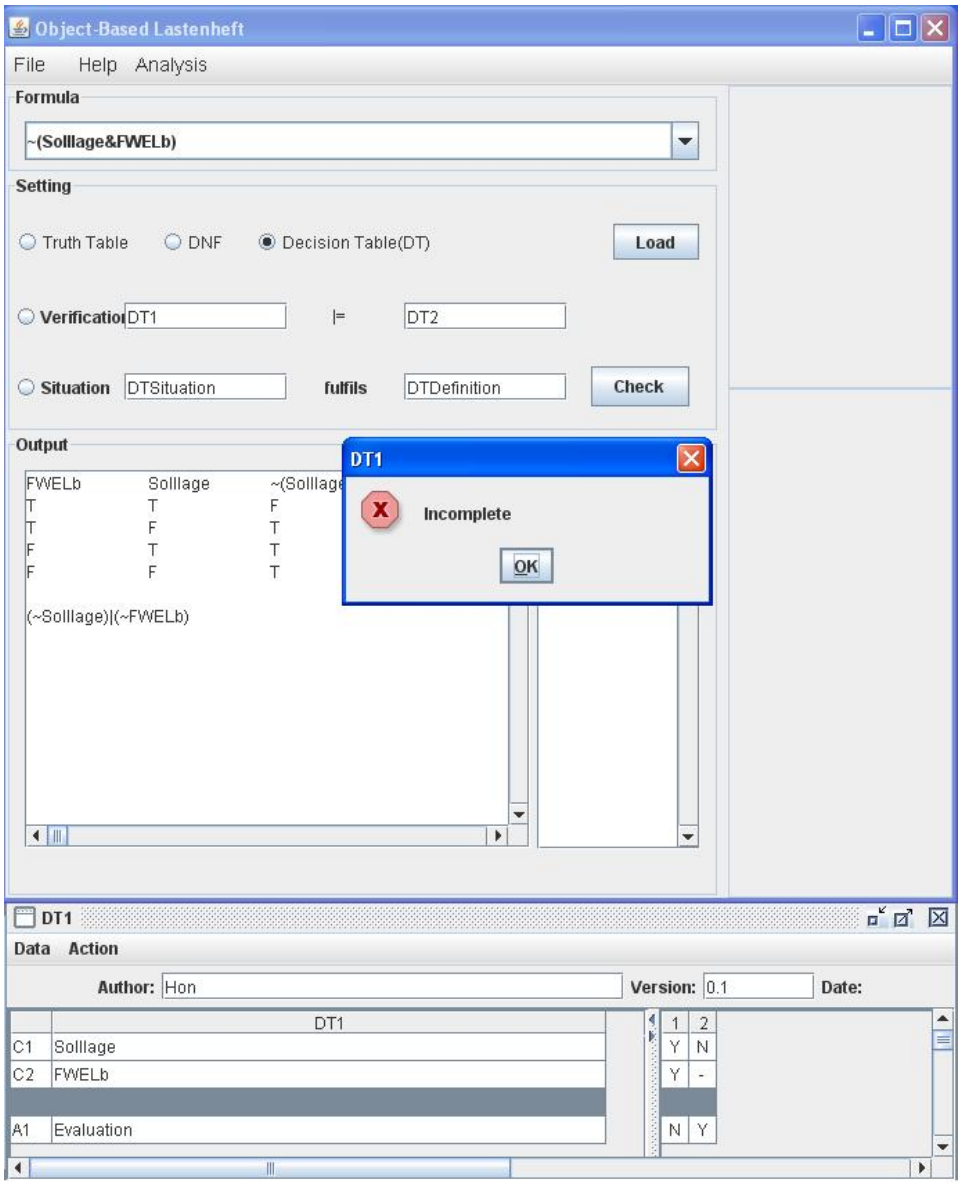


Figure E.9: The result of checking completeness

4. Type in the name of the decision table of ψ (e.g.DTψ) in the right input box of the panel *Verification*.

5. Click the button *Check*.

6. If ψ is a logical consequence of ϕ, a message box *DTϕ implies DTϕ* pops-up. A decision table is generated. It is composed of a single rule with action `Y`.

7. If ψ is not a logical consequence of ϕ, a message box *DTϕ NOT implies DTϕ* pops-up. A decision table is generated. In this decision table, rules with action `N` indicate rules of DTϕ that are not logical consequents to DTψ. In other words, these rules have action `Y` in DTϕ, while they have action `N` in *DTψ* (see Figure E.13).

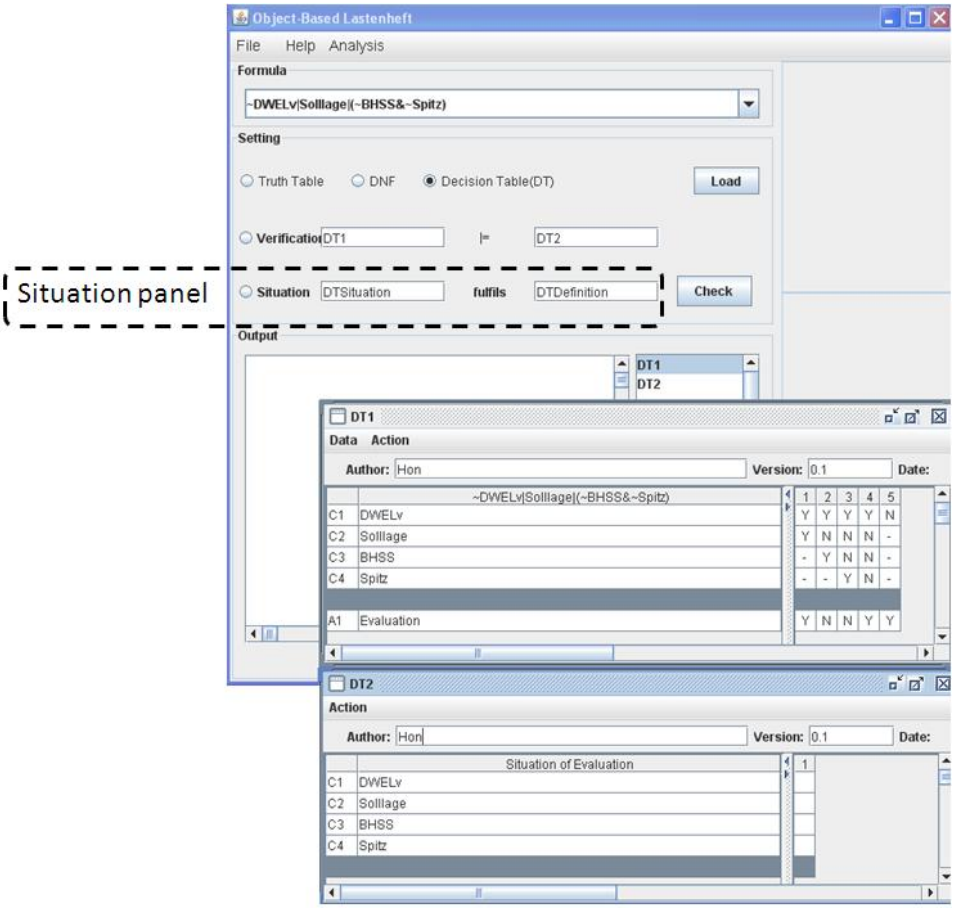


Figure E.10: Specifying a situation

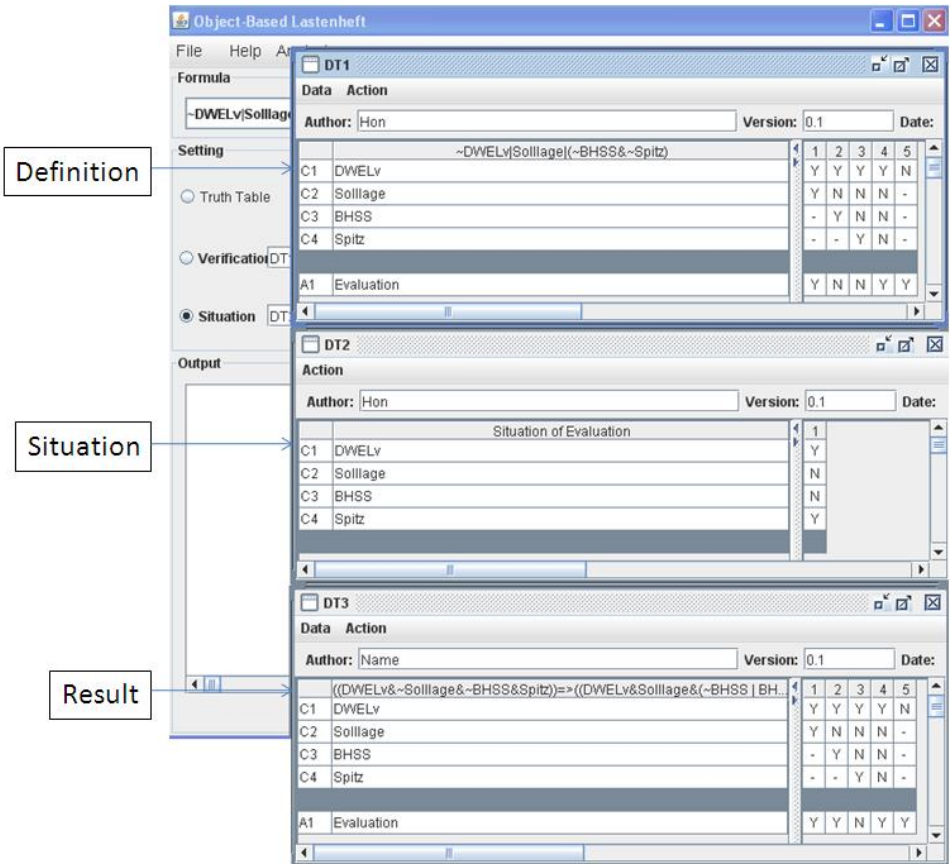


Figure E.11: The result of situational analysis

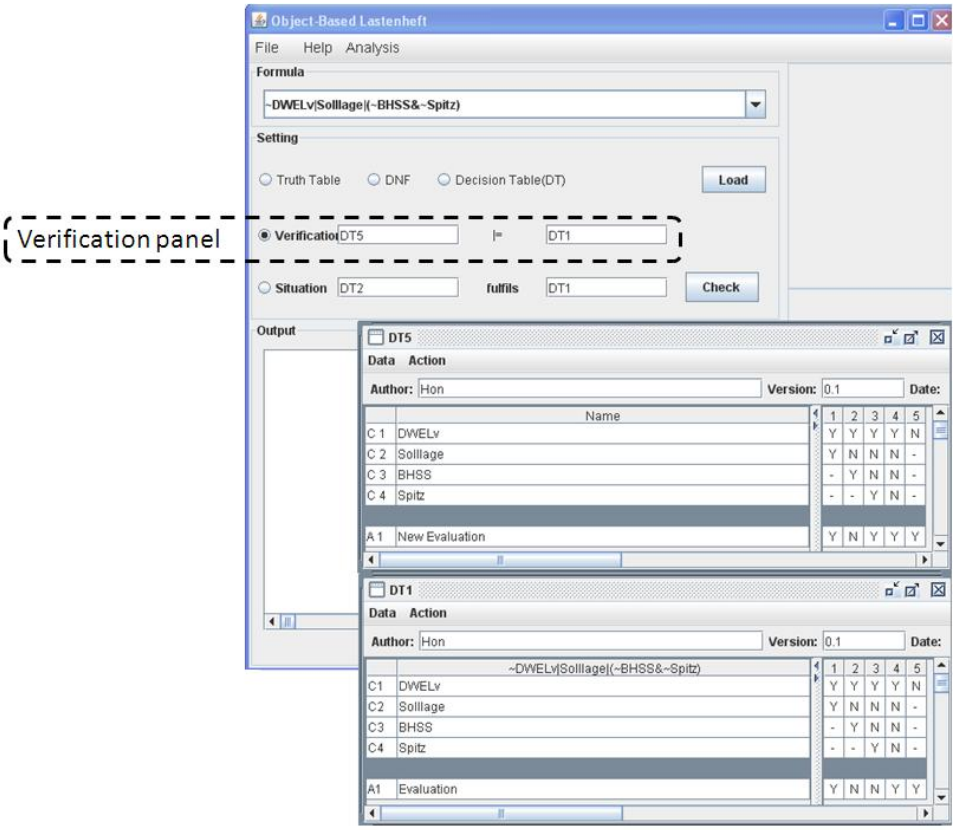


Figure E.12: The verification Panel

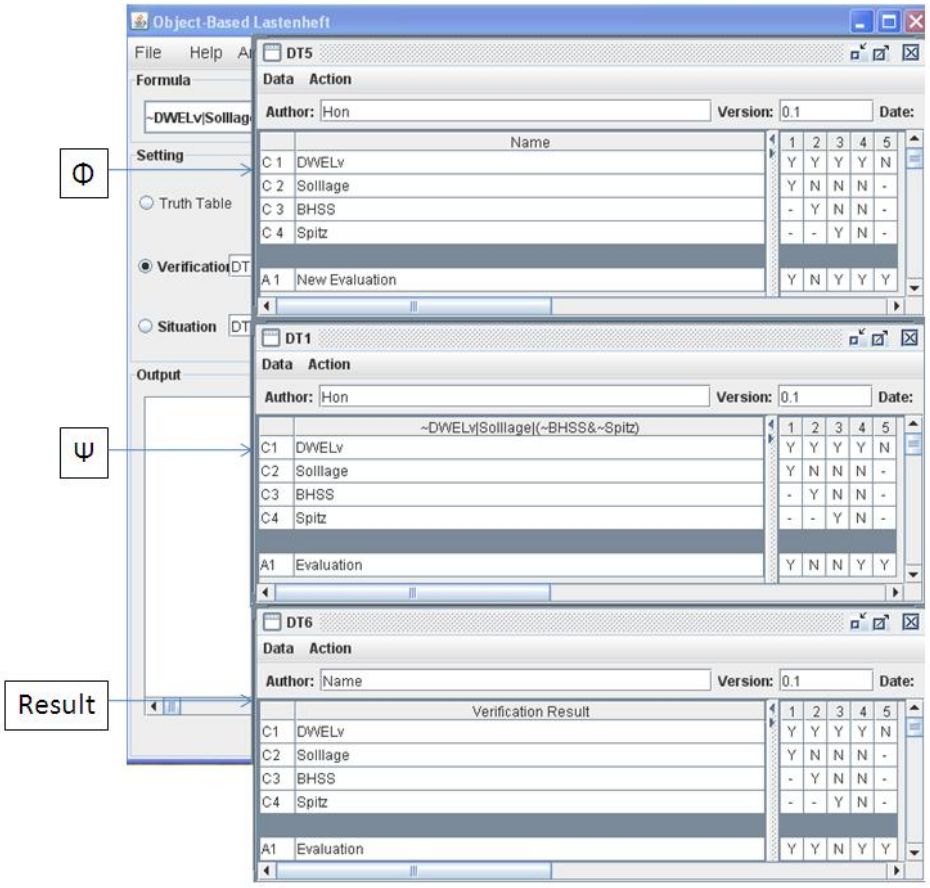


Figure E.13: The result of verification

Bibliography

[Ade08] Adelard. Specbox. `http://www.adelard.com`, 2008.

[And98] H.R Andersen. An introduction to binary decision diagrams. Lecture Notes for 49285 Advanced Algorithm E97, Technical University of Denmark, Department of Information Technology, 1998.

[AP98] V.S. Alagar and K. Periyasamy. *Specification of Software Systems*. Springer, 1998.

[Bae05] J. C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005.

[BC00] R.E Bloomfield and D. Craigen. Formal methods diffusion: Formal methods diffusion, 2000.

[BdRV01] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.

[Bit02] F. Bitsch. Requirements on methods and techniques in perspective of the approval process for railway systems. In *Proceedings of 2nd International Workshop on Integration of Specification Techniques for Applications in Engineering (INT)*, 2002.

[BL99] H.K. Büning and T. Lettmann. Propositional logic: Deduction and algorithms. In *Cambridge Tracts in Theoretical Computer Science*. Cambridge Press, 1999.

[BM02] Randal E. Bryant and C. Meinel. Ordered binary decision diagrams. *Logic Synthesis and Verification*, pages 285–307, 2002.

[Bon01] C. F. Bonnett. *Practical Railway Engineering*. Imperial College Press, London, 2001.

[Bow97] J. Bowen. Formal methods. `http://archive.comlab.ox.ac.uk/comp/formalmethod.html`, 1997.

[BRB90] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a bdd package. In *DAC '90: Proceedings of the 27th ACM/IEEE conference on Design automation*, pages 40–45, New York, NY, USA, 1990. ACM Press.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[BS98] A. Boralv and G. Stalmarck. Models of the railway system infrastructure. In *The First FMERAIL Semainar*, June 1998.

[BW96] B. Bollig and I. Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Comput.*, 45(9):993–1002, 1996.

[CEN99] CENELEC. *EN* 50126 – *Railway Applications - The specification and demonstration of Reliability, Abailability, Maintainability and Safety (RAMS)*, 1999.

[CEN00a] CENELEC. *preEN* 50128 – *Railway applications - Software for railway control and protection systems*, 2000.

[CEN00b] CENELEC. *preEN* 50129 – *Railway applications - Safety related electronic systems for signalling*, 2000.

[CEW93] I. Claßen, H. Ehrig, and D. Wolz. Models for concurrency. In *Algebraic Specification Techniques and Tools for Software Development: The Act Approach*. AMAST Series in Computing, 1993.

[CGM+98] A. Cimatti, F. Giunchiglia, G. Mongardi, D. Romano, F. Torielli, and P. Traverso. Formal verification of a railway interlocking system using model checking. *Formal Aspects of Computing*, 10(4):361–380, 1998.

[CGP99] E. M. Clark, O. Grumberg, and D. Peled. *Model Checking*. The MIT Press, 1999.

[Con79] A.P Conn. Representation of decisions in a requirements specification language. In *In Proceedings of Computer Software and Applications Conference, COMPSAC 79*, pages 113–116, 1979.

[Cor08] B Core. B toolkit. `http://www.b-core.com/ONLINEDOC/BToolkit.html`, 2008.

[DM95] B. Dehbonei and F. Mejia. Formal development of safety-critical software systems in railway signalling. *Applications of Formal Methods*, pages 227–252, 1995.

[Dou99] B. Douglass. UMLstatecharts. `http://www.embedded.com/1999/9901/9901feat1.htm`, 1999.

[Dre02] R. Drechsler. Jade implementation and visualization of a bdd package in java. `http://citeseer.ist.psu.edu/503345.html`, 2002.

[EAF99] L. h. Eriksson, L. Ab, and M. Fahlen. An interlocking specification language. In *In ASPECT IRSE 99, Papers of the International Conference, Institute of Railway Signalling Engineers*, 1999.

[EW00] R. Eshuis and R. Wieringa. Requirements-level semantics for UML statecharts. In *Fourth International Conference on Formal methods for open object-based distributed systems IV*, pages 121–140, Norwell, MA, USA, 2000. Kluwer Academic Publishers.

[FFK88] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of boolean comparision method based on binary decision diagrams. In *International Conference on Computer-Aided Design (Santa Clara, CA)*, pages 2–5, 1988.

[GP95] N. Gorla and H.-C.and Rom W.O. Pu. Evaluation of process tools in systems analysis. *Information and Software Technology*, 37(2):119–126(8), 1995.

[Gro95] The RAISE Method Group. *The RAISE Development Method*. Prentice Hall, 1995.

[Gro97] J.F. Groote. The syntax and semantics of timed μCRL. Technical Report SEN-R9709, CWI Amsterdam, 1997.

[GVVK95] J. F. Groote, S. F. M. Van Vlijmen, and J. W. C. Koorn. The safety guaranteeing system at station hoorn-kersenboogerd. In *Utrecht University*, pages 57–68. IEEE, 1995.

[Hal60] P.R. Halmos. *Naive Set Theory*. Van Nostrand, 1960.

[Han98] K.M Hansen. Formalising railway interlocking systems. In *The Second FMERAIL Semainar*, October 1998.

[Har87] D. Harel. Statecharts: A visual formalism for complex system. *The Science of Computer Programming*, 8:231–274, 1987.

[Har06] J. Harney. Decision tables. `http://www.cems.uwe.ac.uk/jharney/table.html`, 2006.

[HB95] M.G Hinchey and J.P. Bowen. *Applications of Formal Methods*. Prentice Hall, 1995.

[HGE08] Y.M Hon, J.-T. Gayen, and H.-D. Ehrich. Oolh: A formal framework for specifying system requirements. In *SIGSAND-EUROPE*, pages 75–78, 2008.

[HGS09] Y.M. Hon, J.-T. Gayen, and F. Salbert. Formale Spezifikation der Zulassungsprüfung einer Zugstraß unter den Umstellbedingungen für Weichen, Gleissperren und Kreuzungen des Lastenhefts ESTW-R (Regionalstrecken) der Deutschen Bahn AG. Technische Universität Braunschweig, Institut für Eisenbahnwesen und Verkehrssicherung, 2009.

[HH02] P. Hoyningen-Huene. *Formale Logik. Eine philosophische Einführung.* Reclam, 2002.

[HK06] Y. M. Hon and M. Kollmann. Simulation and Verification of UML-based Railway Interlocking. In S. Merz and T. Nipkow, editors, *Preliminary Proceedings of the 6th International Workshop on Automated Verification of Critical Systems*, pages 168–172, September 2006.

[HL96] M. P. E. Heimdahl and N. G. Leveson. Completeness and consistency in hierarchical state-based requirements. *Software Engineering*, 22(6):363–377, 1996.

[Hod77] W. Hodges. *Logic.* Penguin, 1977.

[Hon06] Y.M. Hon. Modeling and verification of UML-based railway interlockings. Master's thesis, Technical University of Braunschweig, Instiute of Information Systems, Germany, 2006.

[HP00] A. E. Haxthausen and J. Peleska. Formal development and verification of a distributed railway control system. *IEEE Transactions on Software Engineering*, 26(8):687–701, 2000.

[HP02] A. E. Haxthausen and J. Peleska. A domain specific language for railway control systems. In *The Sixth Biennial World Conference on Integrated Design and Process Technology, IDPT2002*, pages 168–172, June 23-28 2002.

[HR04] M Huth and M. Ryan. *Logic in Computer Science.* Prentice Hall, 2 edition, 2004.

[Hur82] R.B Hurley. *Decision Tables in Software Engineering.* John Wiley and Sons, Inc., 1982.

[HZ95] D.N. Hoover and Chen Z. 10 th annual conference on computer assurance(compass'95). In *Tablewise, a decision table tool*, pages 97–108. IEEE, June 1995.

[INE09] INESS. INtegrated European Signalling System. `http://www.iness.eu/`, 2009.

[Joh93] R. Johnsonbaugh. *Discrete Mathematics.* Prentice Hall, New Jersey, 1993.

[Jon90] C.B. Jones. *Systematic Software Development using VDM.* Prentice Hall, second edition, 1990.

[Jüt89] G. Jüttner. *Entschiedungstabellen und wissensbas.* Systeme, 1989.

[KE03] N.H König and S. Einer. The euro-interlocking formalized functional requirements approach (eiffra). In *4 th Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'03)*, 2003.

[KM08] T. King and J. Marasco. What is the cost of a requirement error? `http://www.stickymind.com`, 2008.

[Kni98] John C. Knight. Challenges in the utilization of formal methods. In *FTRTFT '98: Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 1–17, London, UK, 1998. Springer-Verlag.

[Koo99] P. Koopman. Verification/validation/certification. `http://www.ece.cmu.edu/koopman/dess99/verification/index.html`, 1999.

[Kos06] V. Kossovoi. Ein editor für entscheidungstabellen mit funktionen zur konsistenzsicherung sowie verbesserungen der benutzerergonomie. Master's thesis, Technische Universität Braunschweig,Institut für Informationssysteme, Deutschland, 2006.

[Lau65] I. P. Lauarance. Conversions of decision tables to computer programs. *Communications of the ACM*, 8:385–390, 1965.

[Lee08] M. Lee. Lecture notes for natural deduction for propositional logic. `http://www.cs.bham.ac.uk/mgl/languageandlogic/lectures/lecture5.pdf`, 2008.

[LSP07] T. Lecomte, T. Servat, and G. Pouzancre. Formal methods in safety-critical railway systems. In *Proceedings of Brazilian Symposium on Formal Methods*, August 2007.

[Mej98] F. Mejia. Formalizing existing safety-critical software. In *The First FMERAIL Semainar*, 1998.

[Mes06] J. Meseguer. From OBJ to maude and beyond. *LNIC*, 4060:252–280, 2006.

[Mil06] R. Miller. Practical UML: A hands-on introduction for developers. `http://bdn.borland.com/article/0,1410,31863,00.html`, 2006.

[ML86] M. Marcotty and H. Ledgard. *The World of Programming Languages.* Springer-Verlag, Berlin, 1986.

[Mon74] M. Montablbano. *Decision Tables.* Science Reasearch Associates, 1974.

[Mon03] J.F Monin. *Understanding Formal Methods.* Springer, 2003.

[Nau60] P. Naur. Revised report on the algorithmic language algol 60. *Communications of the ACM*, 3(5):299–314, 1960.

[omg05] Unified modeling language superstructure specification, v2.0. Object Management Group, 2005.

[Pac04a] J. Pachl. *Railway Operation and Control*. VTD Rail Publishing, New Jersey, 2004.

[Pac04b] J. Pachl. *Systemtechnik des Schienenverkehrs Bahnbetrieb planen, steuern und sichern*. Teubner, Wiesbaden, 4 edition, 2004.

[Pac08] J. Pachl. Glossary of railroad operation and control. `http://joernpachl.gmxhome.de/glossary.htm`, 2008.

[PGHD04] J. Peleska, D. Große, A. E. Haxthausen, and R. Drechsler. Automated verification for train control systems. In *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004)), Braunschweig*, 2004. `http://www2.imm.dtu.dk/pubdb/p.php?3438`.

[Pro07] Community Z Tools Project. Community z tools. `http://czt.sourceforge.net`, 2007.

[Rud93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *ICCAD '93: Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, pages 42–47, Los Alamitos, CA, USA, 1993. IEEE Computer Society Press.

[Sch01] S. Schneider. *The B Method: An Introduction*. Palgrave, 2001.

[Som04] I. Sommerville. *Software Engineering*. Pearson, Boston, 2004. Seventh Edition.

[Spi88] J.M. Spivey. Undertanding z: A formal language and its formal semantics. *Cambridge Tracts in Theoretical Computer Science*, 3, 1988.

[Str77] H. Strunz. *Entscheidungstabellentechnik*. Hanser, 1977.

[Sun09] Sun. JavaCC Project Home. `https://javacc.dev.java.net/`, 2009.

[TE04] C. Trog and L.-H. Eriksson. Spezifikation von stellwerkslogik mit formalen methoden. *Signal+Draht*, 1+2:18–21, 2004.

[The97] T. Theobald. *Transformation Techniques for Decision Diagrams in Computer-Aided Design*. PhD thesis, University of Trier, 1997.

[Tut06] W. Tutschke. *Lastenheft für das Elektronische Stellwerk Regional (ESTW-R), Teilheft F1R Grundausbau (GRU)*. DBNetz, Frankfurt am Main, Deutschland, 2006.

[Van91] J. Vanthienen. Knowledge acquisition and validation using a decision table engineering workbench. In *The World Congress on Expert Systems (WCES)*. Pergamon Press, 1991.

[Van05] J. Vanthienen. Consistency by construction: Decision table experiences in business rules and business processes. In *European Business Rules Conference*, pages 67–74, June 2005.

[Van08] J. Vanthienen. Prologa. `http://www.econ.kuleuven.be/prologa`, 2008.

[VD94a] J. Vanthienen and E. Dries. Illustration of a decision table tool for specifying and implementing knowledge based systems. *International Journal on Artificial Intelligence Tools*, 3:267–288, 1994.

[VD94b] J. Vanthienen and E. Dries. Illustration of a decision table tool for specifying and implementing knowledge based systems. *International Journal on Artificial Intelligence Tools*, 3(2):267–288, 1994.

[VOQ88] W. Van Orman Quine. *Grundzüge der Logik*. Suhrkamp, six edition, 1988.

[Win90] J.M. Wing. A specifier's introduction to formal methods. *IEEE Computer*, 23:8–24, 1990.

[Win02] K. Winter. Model checking railway interlocking systems. In *ACSC '02: Proceedings of the twenty-fifth Australasian conference on Computer science*, pages 303–310, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[Wor08] WorldRailFans. German-english dictionary of railway or technical terms. `http://www.worldrailfans.info/Articles/Europe/Ger-EngDictionary.shtml`, 2008.

[ZB99] H. Zantema and H. Bodlaender. Sizes of decision tables and decision trees. Zantema, H. and Bodlaender, H. L. Sizes of decision tables and decision trees. Tech. Rep. UU-CS-1999-31, Utrecht University, Department of Computer Science, 1999. Available via `http://www.cs.uu.nl/docs/research/publication/TechRep.html`, 1999.